1. Introduction

Background

The Department of Defense (DoD) established the Advanced Distributed Learning (ADL) Initiative in 1997 to develop a DoD-wide strategy for using learning and information technologies to modernize education and training and to promote cooperation between government, academia and business to develop e-learning standardization. The ADL initiative has defined high-level requirements ("-ilities") for learning content, such as content reusability, accessibility, durability and interoperability to leverage existing practices, promote the use of technology-based learning and provide a sound economic basis for investment. The Sharable Content Object Reference Model (SCORM) defines a reference model for sharable learning content objects that meet these high-level

requirements. *The SCORM™ Implementation Guide: A Step-by-Step Approach* describes one way of applying the SCORM to a typical instructional design process.

The ADL Initiative is sponsored by the Office of the Secretary of Defense Deputy Under Secretary of Defense for Readiness (OSD DUSD[R]).

Purpose/Audience

The purpose of *The SCORM™ Implementation Guide: A Step-by-Step Approach* is to serve as practical guidance to instructional designers and developers for implementing the SCORM. The guide illustrates how the SCORM may affect the instructional design and development process of a Web-based training, as well as how the designer or developer could approach course development when tasked with producing a SCORM conformant product that runs on a SCORM-conformant Learning Management System (LMS) or Learning Content Management System (LCMS).

It is not the intent of this guide to teach the instructional design or development process. It is assumed that the reader is an experienced instructional designer or developer who requires information on how the instructional design and development process is altered in order to take advantage of implementing the SCORM. It also is assumed that the designer or developer has a basic knowledge of the SCORM.

It is important to understand that there are many different approaches to designing and developing instructionally sound Web-based content. The authors of this document are providing *one approach* to the process; feel free to take the necessary SCORM guidance and incorporate it into your own approach to designing/developing Web content.

Guide Structure

Although this guide follows a logical sequence of activities, you may read any or all sections of this manual in any order that is appropriate for your knowledge and experience.

For ease of use and presentation purposes, this guide follows a generic, four-phase instructional design and development process as shown in Figure 1. The SCORM can be implemented into any design and development process; it is instructionally neutral and attempts to provide a means for expressing instructional design and development in a manner that is executable by an LMS or LCMS via the Web.

Figure 1. Guide Design and Development Process

As shown in Figure 2, the guide focuses only on the steps in the instructional design and development process that are affected by the SCORM, as highlighted in red. You will see this organizer graphic at the beginning of each section of the guide; with the appropriate section highlighted, tracking where you are in the instructional design and development process.

Figure 2. Guide Organizer Graphic

Scope

There are many different approaches to developing SCORM-conformant content; this manual is used as an example approach, which is not the only approach to implementing the SCORM. Hypertext Markup Language (HTML) and Extensible Markup Language (XML) are used as examples throughout this guide. It is not in the scope of this document to provide examples of how to develop/design content in authoring tools or tools that generate SCORM-conformant content or to provide guidance for using content repositories.

Organization

The core vocabulary section provides the SCORM terminology that you should study before continuing to other sections. Each section of the guide is organized to answer "What," "Why" and "How."

- The "What" provides a brief definition of a step in the instructional design process to build a context for discussing SCORM implementation.
- The "Why" explains why you should complete that section.
- The "How" provides you with a step-by-step process to implement the SCORM.

This manual is not intended to stand alone; you also will be directed to reference other ADL and SCORM documents (see Section 15 – List of References). You can find these documents, as well as more in-depth information such as the history and background of ADL and the SCORM, on the ADL Web site: http://www.adlnet.org. You should have the following documents available for reference:

- Advanced Distributed Learning, *SCORM™ Version 1.2,* available at http://www.adlnet.org.
- Advanced Distributed Learning, *SCORM™ Version 1.2 Conformance Requirements* Version 1.2, February 15, 2001.

This guide is based on the SCORM Version 1.2 and will continue to evolve with subsequent versions of the SCORM.

Disclaimers

Neither ADL nor this guide recommend or endorse products or manufacturers listed in this document. Products listed may be registered copyrights or trademarks of their respective companies or manufacturers.

Contributors

The following organizations and individuals contributed to the content of this guide:

Air Force Institute for Advanced Distributed Learning
Dr. Jerry A. Boling

U.S. Army Training Support Center
Jack Fedder
Carlton P. Hardy
Juanita G. Winstead

U.S. Naval Education and Training Professional Development and Technology Center
Dennis Knott

Internal Revenue Service
Claude Mathews

Academic Advanced Distributed Learning Co-Laboratory
Judy Brown
John Toews

Advanced Distributed Learning Co-Laboratory
Philip Dodds
Jeff Falls
Nicole Franklin
Anne Hensel
Susan Herald
Alan Hoberney
Paul Jesukiewicz
Linda Monzo
Aimee Norwood
Jonathan Poltrack
Eric Roberts
Somer Ruga
Stacey Smith
Nancy Teich
Schawn Thropp
Ted Townsend

Advanced Distributed Learning Co-Laboratory
Kristy Murray
Amy Rossmark

Key Contributors
Concurrent Technologies Corporation (*CTC*)
Institute for Defense Analyses (IDA)
Joint Submarine Analysis Group (JSAG)
Others TBD

Special thanks to the Office of the Secretary of Defense for their sponsorship of the ADL Initiative and Dr. Robert Wisher, Director, ADL.

Point of Contact

If you have questions, concerns or comments regarding SCORM implementation, visit the ADLNet Web site (http://www.adlnet.org) Help & Info Center or contact:
Advanced Distributed Learning Co-Laboratory
1901 N. Beauregard Street
Suite 600
Alexandria, Virginia 22311
703-575-2000

2. SCORM Core Vocabulary

SCORM Core Vocabulary identifies terminology that the user should be familiar with to benefit fully from the sections that follow. Refer to these terms as needed throughout the SCORM implementation process or consult the SCORM for more detailed definitions.

Asset

An Asset is a single electronic file representing media, text, image, audio, etc. Assets can include, but are not limited to, jpg, txt, wav, avi, html and gif file types.

Sharable Content Object (SCO)

A SCO is a collection of one or more assets that is capable of being delivered via a Web browser and has the capability of communicating to a Learning Management System (LMS) via the SCORM Run-Time Environment (RTE) Application Program Interface (API). Currently, SCOs may not link to one another or access data from each other.

Meta-data

Meta-data is "data about data." From a SCORM perspective, meta-data represents a mapping and recommended usage of Institute of Electrical and Electronics Engineers, Inc. Learning Technology Standards Committee (IEEE LTSC) Learning Object Metadata (LOM) to the Content Aggregation Model components (i.e., Assets, SCOs and Content Aggregations). This meta-data is used to describe the Asset, SCO or Content Aggregation in a consistent and meaningful way to enable the search and discovery of the individual components through a content repository.

Content Package

A content package represents a unit of usable and reusable content. A content package includes physical files that may contain a course or some portion of a course. In addition to the physical files and the manifest, the content package also contains control files, and is the mechanism that binds SCORM content model components to the respective meta-data.

Manifest

A manifest is a mandatory XML file that describes the components of a content package, much like a "packing slip." The manifest consists of the following section, the:

- Meta-data section describes the package as a whole
- Organizations section describes one or more hierarchical organizations of the content (content structure)

- Resources section references the actual resource and media files necessary for the content.

Package Interchange File (PIF)

A PIF is a single archive file that contains all of the elements of the content package. SCORM supports the PKZip v.2.04 (.zip) format for PIF files. PIFs are intended to facilitate moving content packages from one system to another.

Data Model

A data model is a common set of information about SCOs that can be tracked by LMSs. From a SCORM RTE perspective, the data model works in conjunction with the API to allow communication between the LMS and the content. The SCORM data model is based on the Aviation Industry CBT (Computer-Based Training) Committee Computer Managed Instruction (AICC CMI) data model defined in the AICC CMI001 "Guidelines for Interoperability" document.[1]

Application Program Interface (API)

The API is a set of predefined functions that is available to a SCO. The SCORM API is based on the AICC CMI API defined in the AICC CMI001 "Guidelines for Interoperability" document.

Learning Management System (LMS)

An LMS is a software application or Web-based technology used to plan, implement and assess a specific learning process. Typically, an LMS provides an instructor with a mechanism to create and deliver content, monitor student participation and assess student performance. An LMS may also provide learners with the ability to use interactive features such as threaded discussions, video conferencing and discussion forums.[2]

Learning Content Management System (LCMS)

An LCMS controls and directs information to a specific learner at a specific time. An LCMS is a multi-developer environment where developers can create, store, reuse, manage and deliver learning content from a central object repository. ADL's focus is on distributed, not central, repositories.

---

[1] *AICC/CMI CM1001 Guidelines for Interoperability* Version 3.4., October 23, 2000. Includes: AICC Course Structure Format, AICC CMI Data Model available at http://www.aicc.org/.

[2] Tech Target, SearchSystemsManagement.com, http://searchsystemsmanagement.techtarget.com/sDefinition/0,,sid20_gci798202,00.html, accessed Oct. 31, 2002.

Advanced Distributed Learning

Business Need

3. Analysis — Identify the Business Need

What

Identifying a need for training is the first step in a uniform method of instructional design. Identifying the business drivers behind the training need ensures a legitimate training issue exists. For example, a training solution for an existing work process is not useful if the performance issue is the result of poor morale; in that case, action to improve morale is more appropriate. However, if educating users about efficient use of the work process increases productivity, a business need for training exists.

Why

In the example above, the decision to implement training would be based on a detailed cost-benefit analysis. The relationship of the SCORM to the business need is significant because of the potential for the SCORM to increase Return On Investment (ROI) over the long term. Implementation of the SCORM may increase the cost benefit of training by allowing for durability, adaptability, interoperability, reusability and scalability of the learning content.

The consideration of technical solutions like SCORM implementation in the early stages of the Analysis Phase may be new to some instructional designers. However, since the implementation decision will affect each phase of the process, it is critical that technical considerations are made early and with significant input from the client.

The question of SCORM implementation is really one of project scope. We know that the client wants effective and instructionally sound Web-based materials. But does the client desire or need to add power to the product by making it durable, adaptable, interoperable, reusable or scalable? If the client desires the type of increased power that SCORM conformance provides, considerations for its implementation should be reflected in the project scope, budget and schedule.

After the decision to implement the SCORM is made, the level of SCORM implementation should be determined. The SCORM Version 1.2 Conformance Requirements Version 1.2 document is downloadable from the ADLNet Web site (http://www.adlnet.org/), and should be used as a guide for defining specific conformance requirements with the client.

The Conformance Requirements document details options for various levels of SCORM implementation. By defining the client's conformance requirements according to the implementation levels defined by the SCORM Version 1.2 Conformance Requirements Version 1.2, you will ensure clear communication of expectations between the client and the content development team. Additionally, by identifying the specific level of SCORM

conformance for SCOs, meta-data and content packages, the development team will be prepared for conformance testing using the SCORM Version 1.2 Conformance Test Suite Version 1.2.2 software. Details of the testing process are presented in the SCORM Verification and Validation section (Section 13) of this document.

How

1. Decide with the client if SCORM is necessary and beneficial.
2. Work with the client to identify the desired level of SCORM conformance, and document the results of the client meeting for later reference during SCORM conformance testing using the SCORM Version 1.2 Conformance Test Suite Version 1.2.2 (See Job Aide 3-1).
3. Reflect decisions regarding SCORM implementation level in project scope, budget and schedule.
4. Keep documentation of lessons learned for further SCORM development.

Job Aide 3-1 — Recording Desired Level of SCORM Conformance

Adapted from the SCORM Version 1.2 Conformance Requirements

Directions: Select one conformance category for each area of conformance: SCO, meta-data and content package.

SCO	
Desired Conformance	**Description**
SCORM Version 1.2 Run-Time Environment Conformant – Minimum	Requirements Summary: The SCO: • Can be launched by a known conformant LMS as defined in Section 3.2 of the SCORM Run-Time Environment and • Searches for and finds an API Adapter as a Document Object Model (DOM) object and • Invokes, at a minimum, the LMSInitialize() and LMSFinish() API functions as described in Section 3.3 of the SCORM Run-Time Environment and • Any additional API functions that are invoked are called correctly.
SCORM Version 1.2 Run-Time Environment Conformant – Minimum with Some Mandatory Data Model Elements	Requirements Summary: The SCO: • Is "SCORM Version 1.2 Run-Time Environment Conformant – Minimum," and • Implements support for correctly getting and/or setting one or more LMS mandatory SCORM Version 1.2 Run-Time Environment Data Model Elements. (Note: LMS mandatory is defined as those data model elements that are required to be implemented by an LMS.) ***Note:*** *If the SCO* ***incorrectly*** *implements one or more mandatory SCORM Version 1.2 Run-Time Environment Data Model Elements, the SCO is non-conformant.*

SCO	
Desired Conformance	**Description**
SCORM Version 1.2 Run-Time Environment Conformant – Minimum with Some Optional Data Model Elements	Requirements Summary: The SCO: • Is "SCORM Version 1.2 Run-Time Environment Conformant – Minimum," and • Implements support for correctly getting and/or setting one or more LMS optional SCORM Version 1.2 Run-Time Environment Data Model Elements. (Note: LMS optional is defined as those data model elements that are optional for implementation by an LMS.) ***Note:*** *If the SCO* ***incorrectly*** *implements one or more optional SCORM Version 1.2 Run-Time Environment Data Model Elements, the SCO is non-conformant.*
SCORM Version 1.2 Run-Time Environment Conformant – Minimum with Some Optional and Some Mandatory Data Model Elements	Requirements Summary: The SCO is: • "SCORM Version 1.2 Run-Time Environment Conformant – Minimum with Some Mandatory Data Model Elements," and • "SCORM Version 1.2 Run-Time Environment Conformant – Minimum with Some Optional Data Model Elements."
None	Content will not be SCORM-conformant.

Meta-Data	
Desired Conformance	**Description**
SCORM Version 1.2 Meta-data XML Conformant – Minimum	Requirements Summary: Content Aggregation, SCO or Asset Meta-data XML Instance: • Is a well formed XML Document and • Is valid against the IMS Learning Resource Metadata Version 1.2.1 XML Schema Definition (XSD) and • Contains elements that conform to their corresponding specified data types and • Contains all mandatory document elements for the corresponding meta-data application profile (Content Aggregation, SCO or Asset) as described in Section 2.2 of the Content Aggregation Model and • Elements defined as having restricted vocabularies adhere to all defined vocabularies as defined in Section 2.2 of the Content Aggregation Model.
SCORM Version 1.2 Meta-data XML Conformant – Minimum with Optional Elements	Requirements Summary: Content Aggregation, SCO or Asset Meta-data XML Instance: • Is "SCORM Version 1.2 Meta-data XML Conformant – Minimum," and • Contains one or more elements that are designated as optional meta-data elements for the corresponding meta-data application profile (Content Aggregation, SCO or Asset) as described in Section 2.2 of the Content Aggregation Model, except for extensions. ***Note**: If the meta-data instance **incorrectly** implements one or more elements that are designated as optional document elements for the corresponding meta-data application profile (Content Aggregation, SCO or Asset) as described in Section 2.2 of the Content Aggregation Model, the meta-data Instance is non-conformant.*

Meta-Data	
Desired Conformance	**Description**
SCORM Version 1.2 Meta-data XML Conformant – Minimum with Extensions	Requirements Summary: The Content Aggregation, SCO or Asset Meta-data XML Instance: • Is "SCORM Version 1.2 Meta-data XML Conformant – Minimum," and • Contains one or more extensions. The extensions used must be well-formed and valid according to the corresponding vendor-provided XML Schema Definition (XSD).
SCORM Version 1.2 Meta-data XML Conformant – Minimum with Optional Elements and Extensions	Requirements Summary: The Content Aggregation, SCO or Asset Meta-data XML Instance: • Is "SCORM Version 1.2 Meta-data XML Conformant – Minimum with Optional Elements," and • Is "SCORM Version 1.2 Meta-data XML Conformant – Minimum with Extensions."
None	SCORM meta-data will not be used.

Content Package	
Desired Conformance	**Description**
SCORM Version 1.2 Content Packaging XML Conformant	Requirements Summary: The Content Package: • If contained in a Package Interchange File (PIF), then the PIF is compatible with PKZIP Version 2.04g and • The Manifest is placed at the root of the Package (e.g., ZIP archive or CD-ROM) and • The Manifest is named "imsmanifest.xml" and • All supporting control documents are placed at the root of the PIF or root directory and • The "imsmanifest.xml" is well-formed XML and • The "imsmanifest.xml" validates against the IMS Content Packaging XML Schema Definition (XSD) Version 1.1.2 and • The "imsmanifest.xml" validates against the ADL Content Packaging XML Schema Definition (XSD) Version 1.2 and • The Content Package contains at least one SCO or Asset (as defined in the SCORM Content Aggregation Model) and • All SCO learning resources identified in the "imsmanifest.xml" are at a minimum SCO-RTE1 and • All meta-data used with the "imsmanifest.xml" adheres to the appropriate SCORM Meta-data Application Profile requirements.
None	Content packages will not be created.

Advanced Distributed Learning

Learner Analysis

4. Analysis – Learner Analysis

What

The learner analysis determines relevant characteristics of the target population. The learner analysis often includes an examination of prior knowledge, skills and attitudes toward the content to be taught.

Why

A learner analysis ensures that the training or instruction is tailored to the needs, abilities and preferences of the target audience and to the specific requirements of adult learners.[1] Traditional learner analysis is moderately affected by the SCORM.

As you conduct your learner analysis, keep in mind that a clear distinction must be made between primary and secondary audiences. Reusability poses a significant challenge. If the goal in the creation of SCORM-conformant content is to reuse the same content with multiple audiences, how do you design for all of those audiences and maintain instructional integrity? The ADL community has identified various mechanisms for addressing this issue. One potential solution is presented.

How

Job Aide 4-1 — Determining Audience Characteristics

1. Work with the client to identify characteristics of the primary audience.
2. Record primary audience characteristics to be used later when:
   - Breaking down content into SCOs.
   - Assigning meta-data to assets, SCOs and content aggregations (See Job Aide 4-2).
3. Work with the client to identify characteristics of the secondary audience.
4. Record secondary audience characteristics to be used later when:
   - Assigning meta-data to assets, SCOs and content aggregations (See Job Aide 4-2).
5. After learner analysis is completed and relevant audience information is recorded, consider the additional actions detailed in Job Aide 4-3.

With this approach, characteristics of the *primary* audience are considered in the content design and meta-data. Content is designed for a specific audience and can be located easily (e.g. in a content repository) and used for similar audiences. This approach also allows the meta-data to reflect potential *secondary* audiences but, in order to maintain

[1] Darryl L. Sink and Associates, Inc., *The Instructional Developer Workshop Participant's Binder*, 2001, page 44.

instructional integrity, does not consider those secondary audiences specifically in the design and breakdown of the content.

Job Aide 4-2 — Meta-data Requirements for Reflecting Primary and Secondary Audiences

Adapted from SCORM Version 1.2

Directions: Record primary and secondary audience characteristics.

Note: The meta-data included in this exercise may be mandatory or optional for Content Aggregations, SCOs and Assets. Refer to the SCORM Version 1.2, Section 2.2.4.4 for additional detail.

Audience Characteristic	SCORM Meta-data Name	Explanation
General Descriptors	Keyword	Keywords or phrases that describe the primary audience.
Culture, geographic location or region	Coverage	The span or extent of such things as time, culture, geography or region that apply to the primary audience.
Age	Typical Age Range	Refers to the developmental age, if it is different from the chronological age.
Language	Language	The primary human language used by the typical end user of the resource.

Job Aide 4-3 — Additional SCORM Considerations During Learner Analysis

If	Then	Refer to
Learner analysis reveals that the learner may be overwhelmed by large amounts of complex content.	Consider learning style when deciding on the size and complexity of SCOs.	Section 6
Learner analysis reveals need for specific content sequencing approach.	Select sequential, non-sequential, user-directed, learner-centric or adaptive sequencing approach, depending on the capabilities of the learning management system.	Section 6
Learner analysis reveals the need for prerequisites.	Ensure that prerequisites are defined in the SCORM meta-data.	Section 9
Instruction may interrupt other job-related activities and the learner may need to leave the training and return later.	Use the SCORM Data Model to periodically record the position where the learner stopped.	Section 7

Advanced Distributed Learning

Context Analysis

5. Analysis – Context Analysis

What

Context analysis is the systematic identification of those characteristics of the training environment that are important to the design of instruction.[1] It is during context analysis that instructional designers seek answers to questions about the size of the audience, who will deliver the training, where and how frequently the training will be delivered and what secondary uses exist for the training content.

Why

A context analysis is performed to identify and describe environmental factors that will affect the design of your training and instruction. Just as the learner analysis is intended to help target the training to a specific audience, the context analysis is intended to target the training for delivery in a specific learning environment.

There has been much debate about whether instruction can be designed with multiple contexts or environments in mind. Traditional instructional design approaches focus on analyzing and building instruction around a specific context. The SCORM introduces the concept of learning object reusability and therefore introduces debates about definition of instructional context. The SCORM introduces a reusability paradox: the most reusable objects are context-independent, while the best instruction is highly contextualized. While the debate of multiple contexts continues, you can incorporate SCORM into your context analysis by reflecting immediate and potential future contexts in SCORM meta-data. Use Job Aide 5-1 as a guide to the use of SCORM meta-data for reflecting context.

[1] Ibid., Sink, page 69.

How

Job Aide 5-1 — Perform Context Analysis and Reflect Results in SCORM Meta-data

If	Then
The context is closely tied to the instructional content.	Reflect the context within which the learning is to take place in SCORM meta-data elements as described in Job Aide 5-2.
There are potential secondary uses for the instruction.	Reflect the primary context for which the instruction was designed by using Job Aide 5-2. Reflect potential secondary audiences by adding descriptive information to the SCORM meta-data identified in Job Aide 5-3.

Job Aide 5-2 — Reflecting Context in SCORM Meta-data

Adapted from SCORM Version 1.2

Directions: Research each type of meta-data using the SCORM Meta-data Information Model, SCORM Version 1.2. Record contextual information relevant to the instructional content and keep this information for use when assigning meta-data to assets, SCOs and content aggregations.

Context Characteristic	SCORM Meta-data Name	Description
Delivery Environment	Context	The principal environment within which the learning and use of the resource is intended to take place.
Learning Time	Typical Learning Time	Approximate or typical time it takes to work through the resource.
Resource Use	Description	Comments on how this resource is to be used (e.g., instructor guidelines that come with a textbook).
Cost	Cost	"Yes" or "No" to the question of whether this instructional content

		requires payment by the user.
Copyright and Other Restrictions	Copyright and Other Restrictions	"Yes" or "No" to the question of whether copyright or other restrictions apply to the use of this resource.

Job Aide 5-3 — Reflecting Secondary Uses in SCORM Meta-data

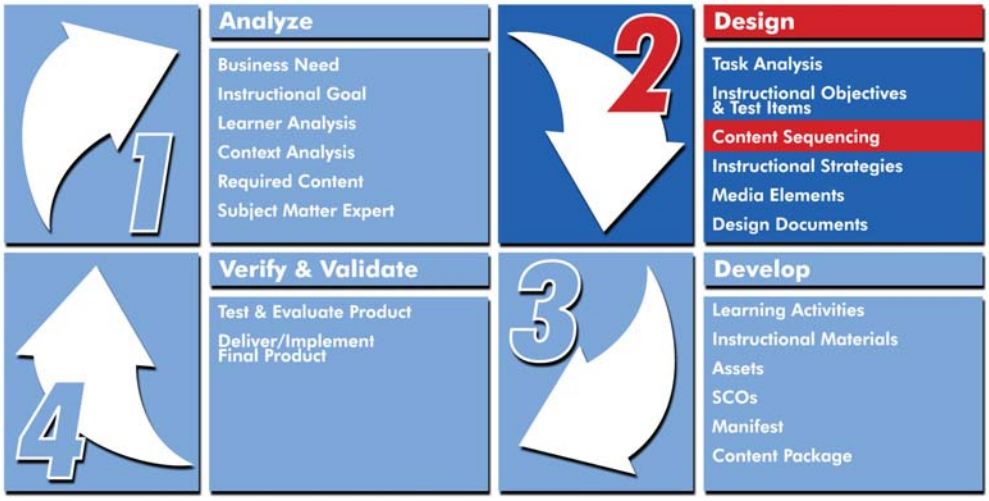
Adapted from SCORM Version 1.2

Directions: Research each type of meta-data using the SCORM Meta-data Information Model, SCORM Version 1.2. Record information about secondary uses for the instructional content using each type of meta-data listed below. Because there is no clearly identified place for reflecting secondary uses within SCORM meta-data, a combination of the meta-data listed here is recommended.

Context Characteristic	SCORM Meta-data Name	Description
Secondary Uses	Description	A textual description of the content of the resource being described.
	Keyword	Keywords or phrases describing secondary uses for this resource.

Advanced Distributed Learning

Content Sequencing



6. Design – Content Sequencing

What

Sequencing is the efficient ordering of content in a way that helps the learner achieve the objectives. Several sequencing strategies may be used within the same training—possibly a different one for each objective, if applicable.

Why

Organizing the content according to a specific sequencing schema provides an orderly method for presenting the content that is likely to match the learner's expectations.[1] Sequential learning paths are simple and reliable; they let the designer control the order of experiences encountered by the learner.[2]

How

When designing your content sequence(s), it is necessary to think in terms of Sharable Content Objects (SCOs). One source of confusion about the size of SCOs is the relatively new idea of "learning objects." There are people in the e-learning community who have strong beliefs about what may constitute a "true" learning object. Some think that a learning object teaches one idea or addresses one single learning objective; others insist that a learning object must include an assessment. Definitions vary widely depending on the learning design theory being applied. However, each situation varies and the approach depends largely on the preferences of the designer and needs of the client.

The SCORM can enable a number of different approaches and can accommodate a variety of design paradigms. For example, one definition of a learning design might include discrete activities such as the following:

- A statement of the objective, introductory material, an overview and/or advanced organizer
- A unit of instruction or an instructional activity (tutorial, simulation or media experience)
- An assessment or series of assessments.

An important part of the design process is deciding how to organize content to create SCOs. Sometimes, deciding what is a SCO is a judgment call; however, it usually is

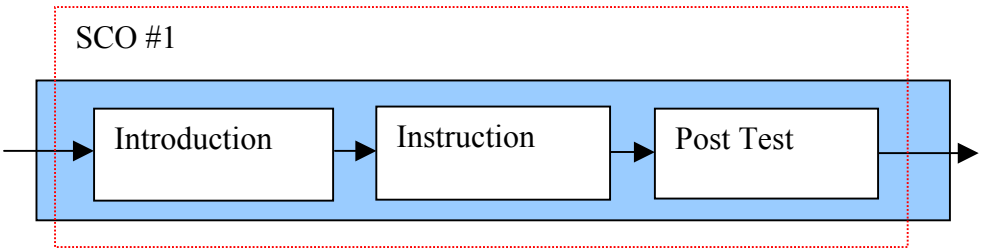
---

[1] Morrison, Ross, and Kemp, *Designing Effective Instruction*, page 120, John Wiley & Sons, Inc., 2001, ISBN: 0-471-38795-9.
[2] William Horton, *Designing Web-Based Training*, page 176, John Wiley & Sons, Inc., 2000, ISBN: 0-471-35614-X.

based on the possible reusability of a SCO (perhaps in the same design) or a need for the LMS to "know" and keep track of the learner. It is possible to create SCOs that include different types or groups of learning activities.

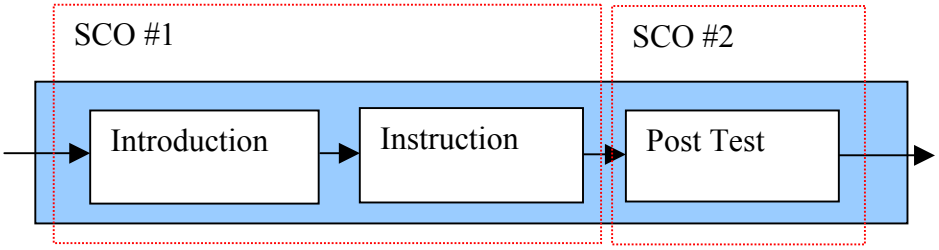
The following possible rationales can guide the process of deciding on the SCO construct to meet the design needs.

SCO Construct #1



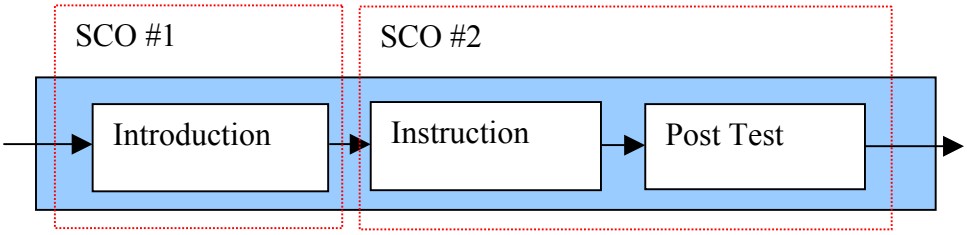
Rationale: The introduction, instruction and post-test activities might be closely related and it would *never* make sense to break them apart and use them elsewhere. Therefore, one SCO is adequate in this situation.

SCO Construct #2



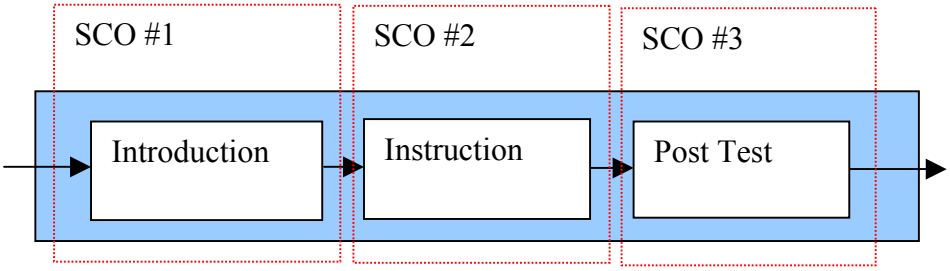
Rationale: The introduction and the instruction are so conceptually intertwined that they are best delivered as one unified activity. The post-test, however, might assess mastery of a particular skill and might be used elsewhere, thus it makes sense to separate it.
In addition, it might be desirable for the Learning Management System (LMS) to "know" when SCO #1 is completed so it can see that the learning material was completed, but the post-test might not have been started. This would be useful if the learner wanted to log out and return later and could begin at SCO #2.

SCO Construct #3



Rationale: Imagine an introductory experience that might contextually relate the instruction and assessment, perhaps using an advanced organizer that directs the learner to pay particular attention to one specific learning objective. Later in the course, the same instruction and assessment (SCO #2) might be introduced with a different learning objective.

SCO Construct #4



Rationale: Each of the activities are separate SCOs that could be reused elsewhere. The advantage to this approach is that the progress through the learning process is known to the LMS, since it tracks where (in which SCO) the learner is and then delivers the next activity.

Another advantage is that later, it becomes easy to exchange SCOs with newer ones or different ones for a slightly different topic. For example, imagine an introduction that reads, "Pay attention to this objective in the instruction you are about to receive." The post-test might be designed only to assess that particular objective. Later in the course, another introduction with a different objective but same instruction can be used and then a different post-test can be delivered for the new objective. Thus, reuse is enabled and the LMS can track the mastery levels of different learning objectives.

It is a good strategy to create SCOs at an individual activity level so that an LMS can keep track of learner progress. Let the goals of your design guide you through the decision process for defining a SCO.

Advanced Distributed Learning

Design Documents

7. Design — Design Documents

What

The design documents are created to provide a framework or blueprint for the course. They can be used to communicate SCORM calls and tracking abilities to the developer. The design documents are used to outline the Sharable Content Objects (SCOs) and assets. For the purpose of this guide, the design documents will be represented as storyboards. Storyboards represent the complete, screen-by-screen layout of the module structure.

Why

Storyboards communicate tracking, content sequencing and objective placement and are an important planning tool for instructional design and SCORM implementation. They will help the designer map out the course and communicate all SCORM calls.

How

The SCORM process will affect the design of the content and storyboards slightly. Here are some items to consider when designing SCORM-conformant content and/or storyboards.

Learning Management Systems

Navigation from SCO to SCO will occur within the Learning Management System (LMS) for SCORM-conformant content. At present, SCORM 1.2 does not mandate navigation schemes. Typically, there are two ways to navigate from one resource to another within an LMS. The first is to use the Table of Contents (TOC) provided by the LMS. The designer can define the titles of the TOC within the manifest file. The second is to use the Next/Previous buttons provided by the LMS to take the user from SCO to SCO. The designer may choose to use a combination of these two approaches or may not have both of these options available, depending on which LMS is being used.

User Interface

Use the following examples and best practices to help you design your user interface to run through an LMS:

- Use a Vector-based background graphic and set the parameters to be scalable instead of fixed
- To assure that buttons on the screen will always be viewable, do not set the buttons in a fixed position

- For testing purposes, develop for the equipment the target audience will be using. If the text can be read, then it is likely a good design for the majority of users.

As shown in Figures 7-1 and 7-2, this particular LMS has a TOC button that will launch the TOC in another window. The learner must select the TOC to navigate from SCO to SCO. To navigate within a SCO page by page the learner uses the Back button (the button on the left) at the bottom of the content. The square button is a rollover image that signifies that the learner is done with the SCO and is placed by the developer to call LMSFinish.

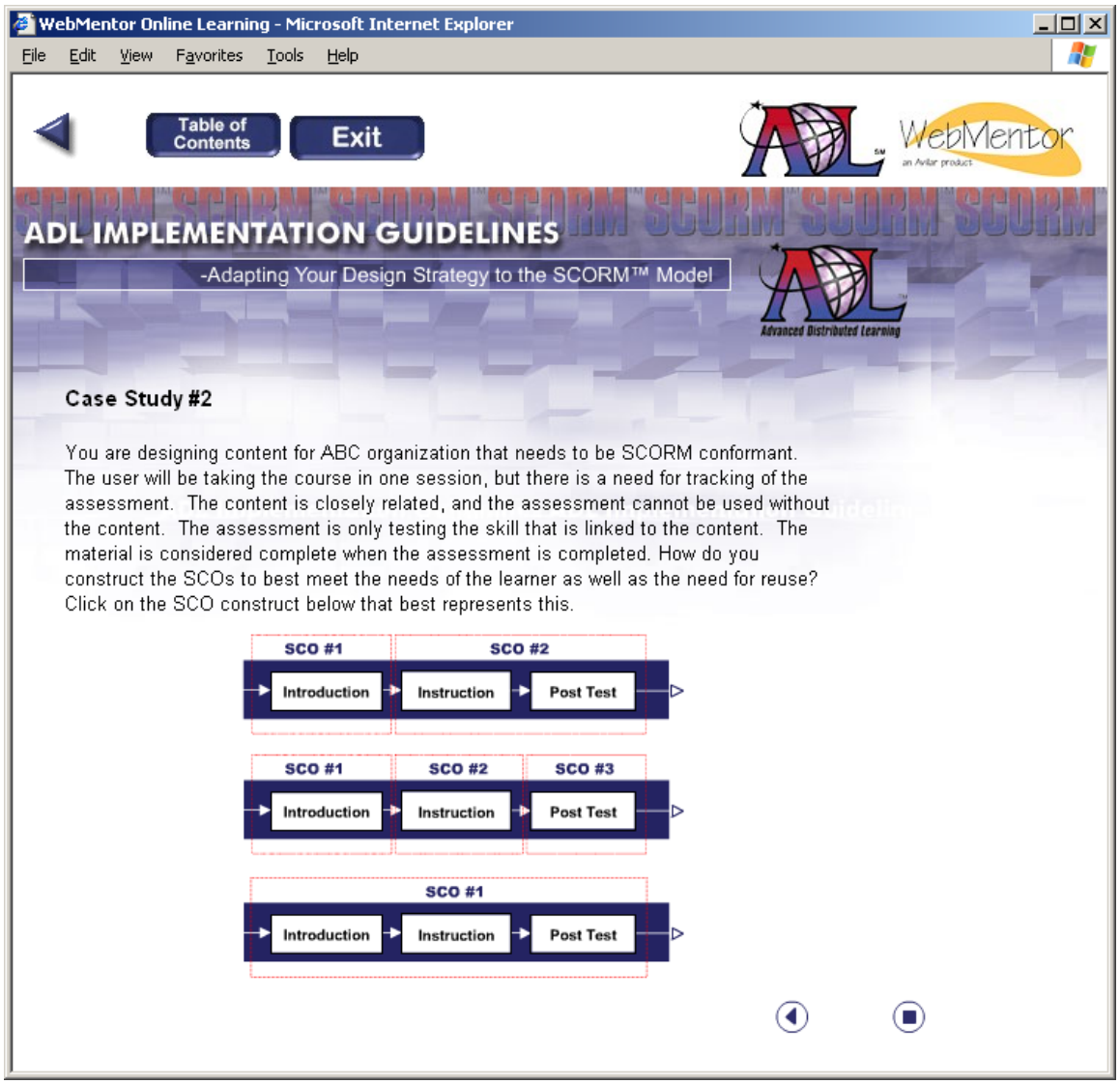


Figure 7-1: LMS With Table of Contents

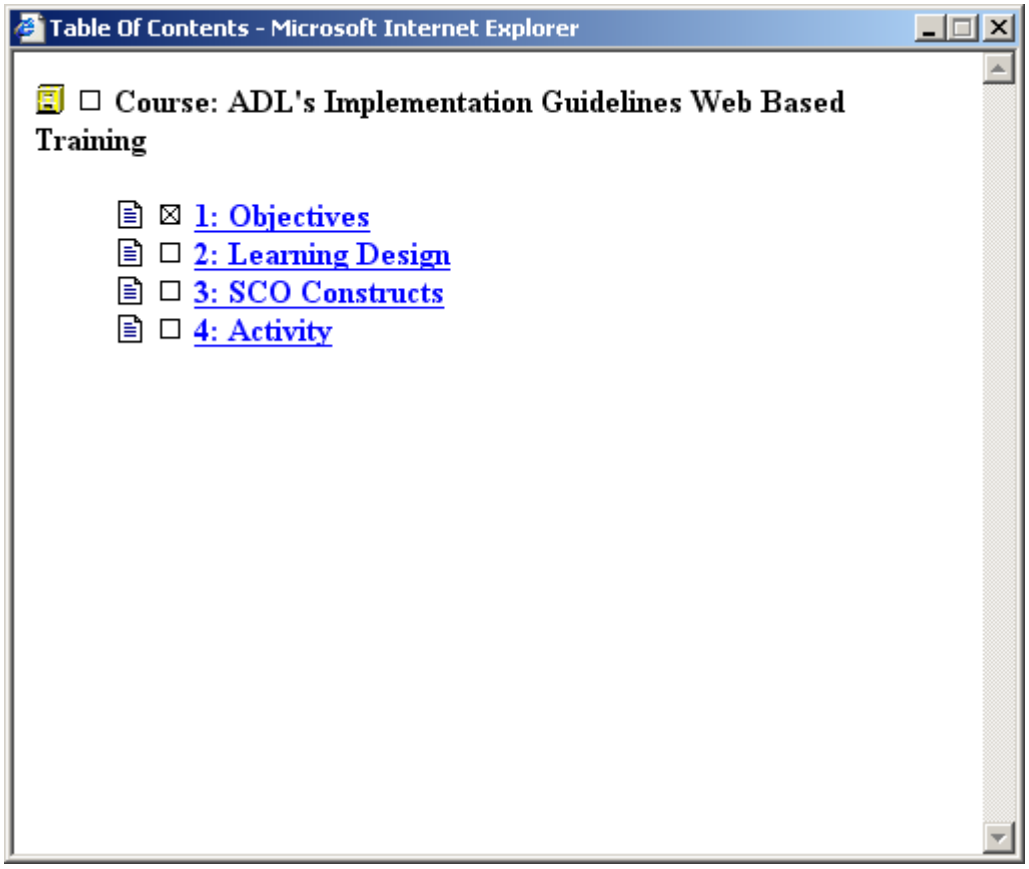


Figure 7-2: Result of Selecting Table of Contents Button

Figure 7-3 shows the ADL Sample Run-Time Environment (RTE) using the choice method provided by the RTE. The learner will navigate from SCO to SCO by using the TOC provided by the RTE (located on the left side of the screen).

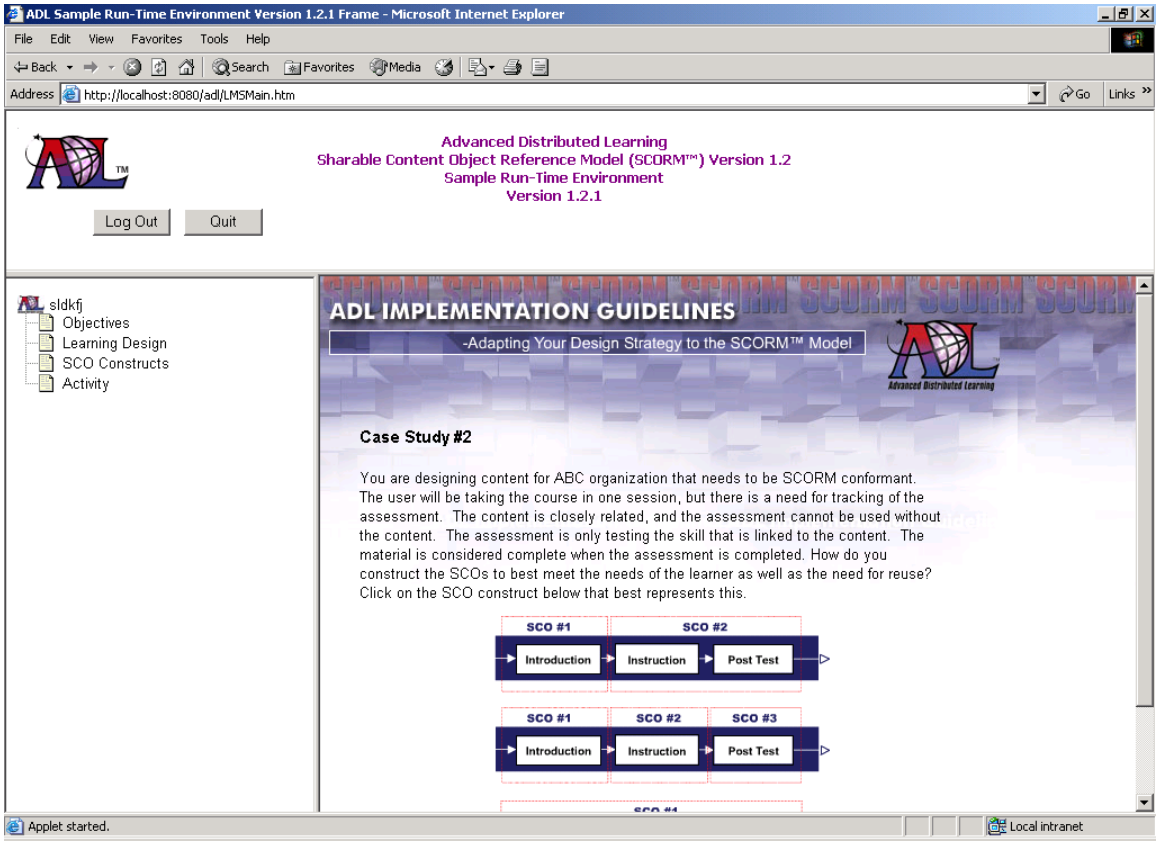


Figure 7-3: Sample RTE Using the Choice Method

Figure 7-4 shows the ADL Sample Run-Time Environment (RTE) using the flow method provided by the RTE. The learner will navigate from SCO to SCO by using the Next/Previous buttons provided by the RTE.

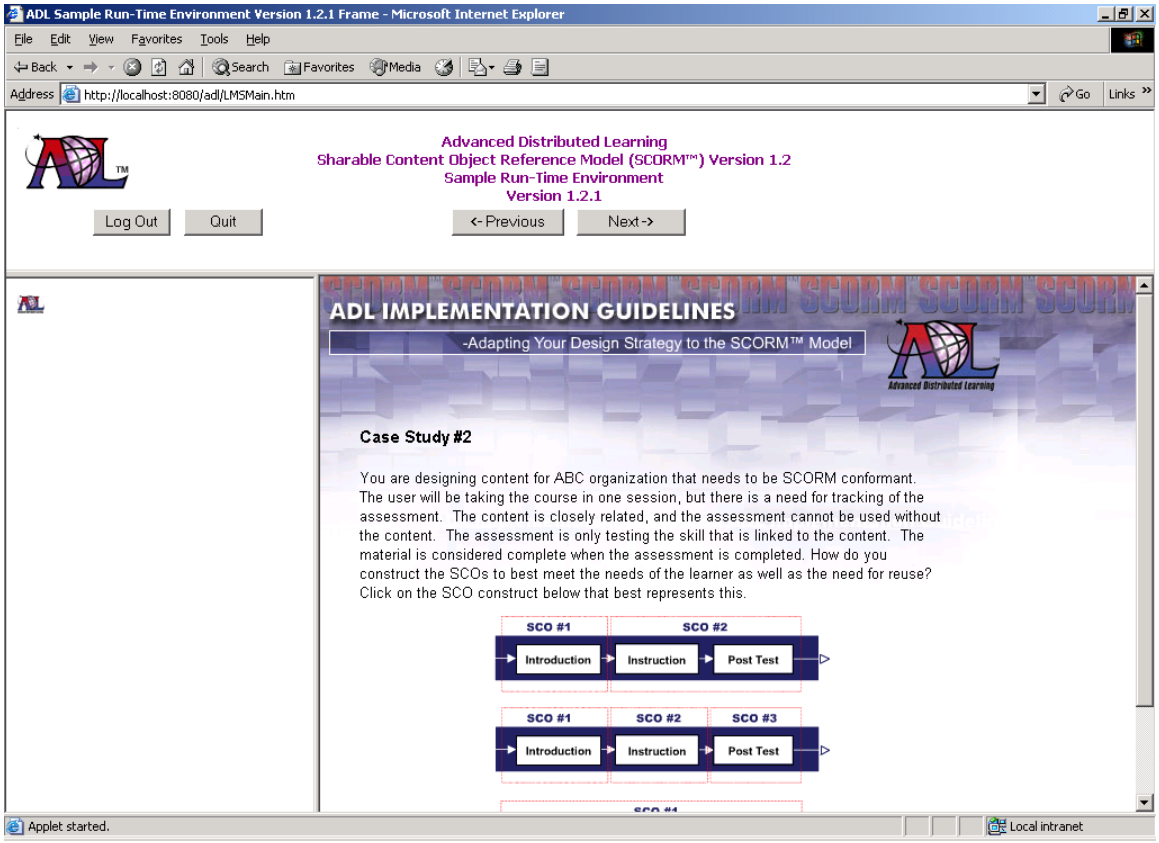


Figure 7-4: Sample RTE Using the Flow Method

Title of SCOs

To promote reusability, avoid titling the SCOs with lesson, module, unit, numbers, etc. Instead, assign titles that reflect the SCO's content. For example, instead of titling the SCO "Lesson 3 – The SCORM," it should be titled simply, "The SCORM."

Acronyms

Tanner, Caudill and Hamel of the University of Central Florida Institute for Simulation and Training and Blickensderfer from Naval Air Systems Command, Training Systems Division share their lessons learned in the article, "Building an Online Course by Developing and Sequencing SCOs." This article documents lessons learned throughout the design and development process of a SCORM-conformant Web-based training. One lesson deals with using acronyms in learning content. The authors state that in order to avoid confusion about what the acronym stands for, it is important to spell out the

acronym at the beginning of each SCO. For example, in SCO 1, the designer defines "SCORM" as the "Sharable Content Object Reference Model," and should do so in SCO 2 as well, with the understanding that the learner may not have seen SCO 1.

Application Program Interface (API)

The API defines a standardized means for SCOs to communicate with the LMS. The LMS does not initiate communications; SCO resources initiate all communication to the API adapter, which in turn communicates with the LMS. The API also facilitates the communication of the data model elements to the LMS. SCOs will communicate with the API via JavaScript calls. JavaScript is the common thread between SCOs and LMSs. Here are some general rules to be used in conjunction with the API:

- Function names are case-sensitive and must be expressed as shown in the SCORM
- Function parameters or arguments are case-sensitive; all parameters should be lower case
- Each call to an API function, other than the error handling functions, resets the error code.

The functions of the API include the following:

- LMSinitialize and LMSfinish – initiates and closes the communication with the LMS
- Get and set values – works with the data model elements to collect and store information by the LMS
- LMS commit – tells the LMS to persist any set data model elements
- LMSgetlasterror LMSgeterrorstring LMSgetdiagnostic – used by the SCO to allow error messages to be viewed when an error has occurred.

Data Model Elements

Data model elements are used to interpret the design and tracking needs into a SCORM call using the API. They provide a common data model to ensure that different LMS environments can track a defined set of information about SCOs. If, for example, it is determined that tracking a learner's score is a general requirement, then it is necessary to establish a common way for content to report scores to the LMS; SCORM helps to accomplish this. If SCOs use their own unique scoring representations, LMSs may not know how to receive, store or process the information. A data model element can be "read," "write" or "read/write." "Read" means that a SCO can only get a value and "write" means that the SCO can set a value. Figure 7-5 shows the data model categories available for use by the SCO.

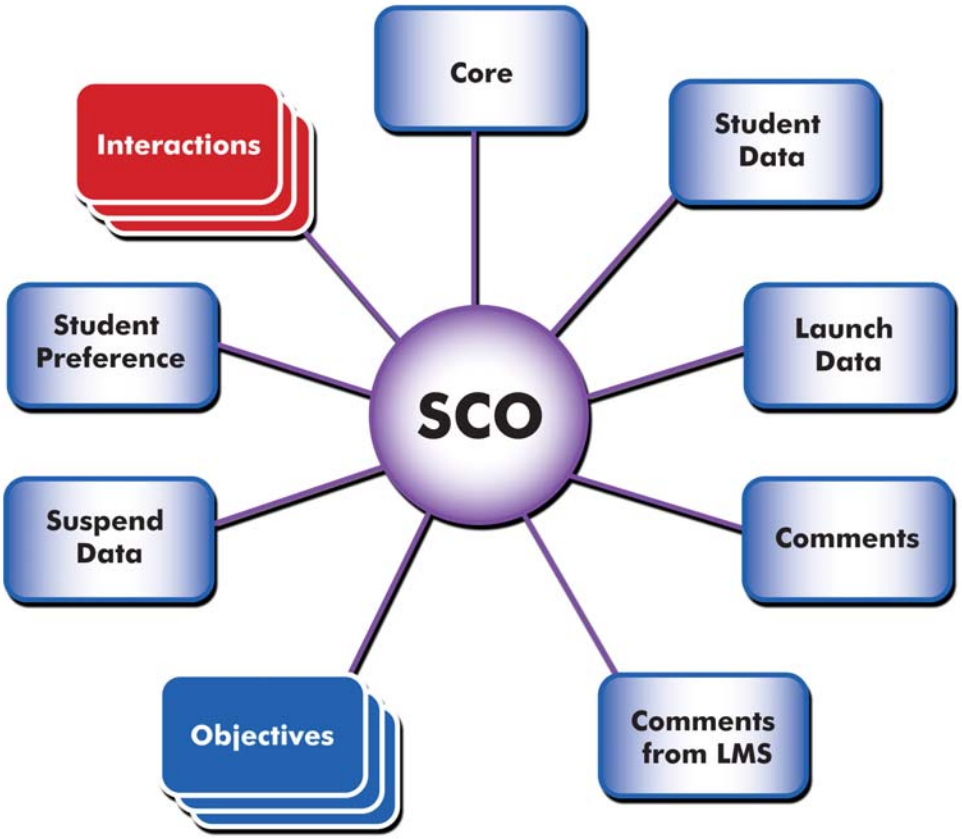


Figure 7-5: Data Model Categories Available for use by the SCO

Job Aide 7-1 — Data Model Elements and Their Uses

Note: This example is just one method in which the data model elements can be used. You can use the elements in any way that you deem necessary, as long as it is supported by the SCORM.

Example Usage	Data Model Elements	Accessibility
Parent: cmi.core – Information required to be furnished by all LMSs. What all SCOs may depend on to start up.		
Gets a list of elements that are supported by the LMS. Determines if the elements that you use are supported by the LMS to get and set values.	cmi.core_children	Read
Gets a unique learner ID for the course that is set by the LMS. Displays the user's ID on the screen throughout the SCO.	cmi.core.student_id	Read
Gets the learner's name. Displays the learner's name on the screen throughout the SCO.	cmi.core.student_name	Read
Provides check point progress through the SCO in order for the learner to return to where they left off in the SCO. This element works well with multiple-page SCOs.	cmi.core.lesson_location	Read/Write
Determines if the SCO is being taken for credit. If the SCO is being taken for credit, the SCO can communicate that status back to the LMS and then make an additional call to set up other elements. If the SCO is not being taken for credit, the SCO can send that information back to the LMS and then turn off additional elements. Commonly used in conjunction with lesson_mode.	cmi.core.credit	Read
Determines the status of the SCO. Determines if the SCO is passed, failed, incomplete, browsed or not attempted. This can be used to lock learners out of a SCO or make them take it again, depending on the status.	cmi.core.lesson_status	Read/Write

Example Usage	Data Model Elements	Accessibility
Determines if it is the first time that the SCO has been launched. If this is not the first time the SCO has been launched, then a welcome back message could be displayed. If it is the first time the learner is visiting the SCO, a different welcome message could be displayed.	cmi.core.entry	Read
Parent: cmi.core.score – indication of the performance of the learner		
Determines if the elements that you are using are supported by the LMS.	cmi.core.score_children	Read
Determines the performance of the learner during the last attempt on the SCO.	cmi.core.score.raw	Read/Write
Determines the maximum score a learner achieved on the SCO. Could be used for testing purposes. For example, there may be a possible 100 points on an exam, so the maximum score would be 100.	cmi.core.score.max	Read/Write
Determines the minimum score that a learner could have received on an exam. Could be used for testing purposes. The minimum score that the learner may receive could be 0.	cmi.core.score.min	Read/Write
Calculates total accumulated time the learner spent in a SCO. Could be used to evaluate the effectiveness of the SCO. For example, if the SCO was supposed to take 10 minutes to complete and the majority of the learners are spending 2 hours in the SCO in conjunction with other criteria, it may be determined that the effectiveness of the content needs to be reviewed.	cmi.core.total_time	Read
Determines the mode the learner was in while in the SCO. *Browse* – previewing material. *Normal* – taking the SCO. *Review* –going through the material again. This element would be a good indicator on	cmi.core.lesson_mode	Read

Example Usage	Data Model Elements	Accessibility
how often the learner visited the SCO and in which mode. Could be used to evaluate the learner's performance.		
Indicates how the learner left the SCO. *Time-out* – the learner ran out of time and the SCO forced them out. *Suspend* – learner left the SCO in order to return to it later. *Logout* – learner left the SCO by logging out of the LMS. Used to let the designer know if there is something that may need to be changed within a SCO. Example: the learner may need additional time to complete the SCO. Used in conjunction with cmi.core.entry.	cmi.core.exit	Write
Determines the time spent in a SCO each time the learner enters that SCO.	cmi.core.session_time	Write
Parent: cmi.suspend_data – information that SCOs set upon premature exit of the SCO, for retrieval upon restart.		
Parent: cmi.launch_data – information represented in the manifest specific to a SCO. Variables a SCO needs to initialize and set up.		
Parent: cmi.comments – mechanism for collecting and distributing comments for a SCO.		
Identifies feedback about the SCO. Asks questions about the SCO. Makes general comments/suggestions about the SCO. Gets learner feedback about the SCO. Used at the end of a SCO for evaluation of the SCO.	cmi.comments	Read/Write
Writes and displays on screen instructor hints/comments/suggestions.	cmi.comments_from_LMS	Read
Parent: cmi.objectives – identifies how a learner has performed on individual objectives within a SCO.		
Determines what is supported by the LMS under cmi.objectives category.	cmi.objectives._children	Read
Determines the current number of objectives within the SCO.	cmi.objectives._count	Read
Identifies the objectives using a specific objective ID. Will use this objective ID to set mastery of the objective and map it to other elements.	cmi.objectives.n.id	Read/Write

Example Usage	Data Model Elements	Accessibility
Parent: cmi.objectives.n.score- each objective can contain an associated score.		
Determines which elements are supported by the LMS.	cmi.objectives.n.score._children	Read
Determines the learner performance after each attempt on an objective. Can be used in conjunction with the interaction elements.	cmi.objectives.n.score.raw	Read/Write
Maximum score that a learner could have received on an objective. Can be used to score a particular objective.	cmi.objectives.n.score.max	Read/Write
Minimum score that a learner could have received on an objective. Can be used to score a particular objective.	cmi.objectives.n.score.min	Read/Write
Determines a status of the objectives. Determines if the learner has passed, failed, completed, not completed, browsed or not attempted an objective. Used in conjunction with interaction elements to link activities to the objective.	cmi.objectives.n.status	Read/Write
Parent: cmi.student_data – information to support customization of a SCO based on a learner's performance.		
Determines the score that needs to be achieved to complete the SCO. Used to determine if there is preset mastery of the SCO.	cmi.student_data.mastery_score	Read
Determines the maximum time a learner is allowed to work in a particular SCO. Defined in the manifest; determines the maximum time the learner is allowed in the SCO.	cmi.student_data.max_time_allowed	Read
When the maximum time that the learner is allowed in a SCO is reached, the designer may set one of the following options: • Learner exits the SCO • Learner continues Displays a message to the learner when the SCO has timed out.	cmi.student_data.time_limit_action	Read

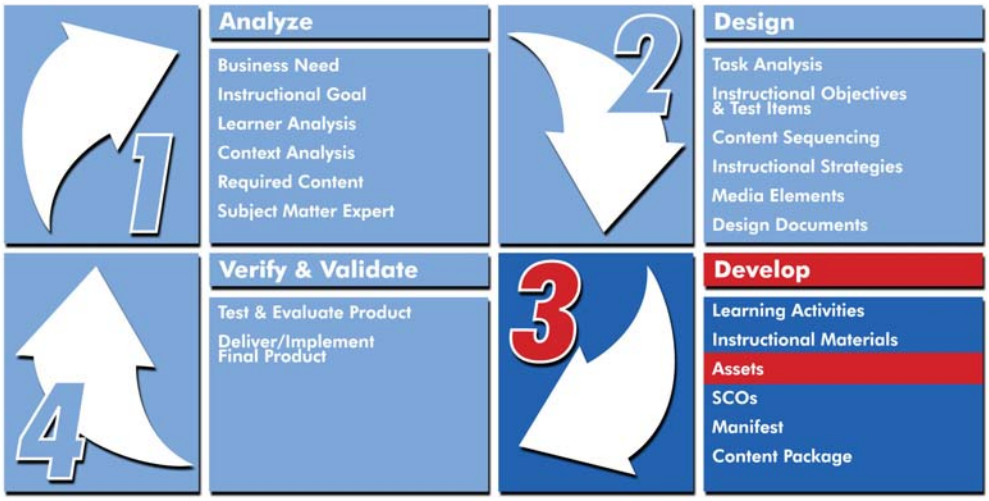
Example Usage	Data Model Elements	Accessibility
Parent: cmi.student_preference – selected options that are appropriate for subsequent SCOs. In each of the following, suspend_data could be used to keep track of the learner's preferences and they then could be used in each SCO.		
Determines which elements are supported by the LMS.	cmi.student_preference._children	Read
Determines whether the audio of a SCO is turned on or off. Sets audio to on or off according to a learner's preference.	cmi.student_preference.audio	Read/Write
Identifies the language in which information should be presented if the SCO has multi-language capability. Identifies and sets the language from the LMS that the learner prefers.	cmi.student_preference_language	Read/Write
Allows learner to change the pace of the content (example: text, simulation, video, voice over, etc.).	cmi.student_preference.speed	Read/Write
Turns off the audio and display it in a text window, if the learner prefers. Leaves the audio on and request that the text be presented simultaneously with the audio, if the learner prefers. Makes the text disappear so that only the audio and graphics are available, if the learner prefers.	cmi.student_preference.text	Read/Write
Parent: cmi.interactions – is a recognized and recordable input of group of inputs from the learner to the computer.		
Determines which elements are supported by the LMS.	cmi.interactions._children	Read
Evaluates how many times a learner has gone to a specific interaction. This can be used for evaluation purposes. If a learner keeps going back to a particular interaction, they may need remediation in this area.	cmi.interactions._count	Read
Identifies an interaction using a unique identifier.	cmi.interactions.n.id	Write
Parent: cmi.interactions.n.objectives – interaction data models will be very effective for keeping detailed statistics on evaluations and exam questions. The interaction data will help the designer determine which questions will need additional rework from a formative and summative evaluation.		

Example Usage	Data Model Elements	Accessibility
SCO will determine the number of objectives set. Used to determine which objective ID records to set in correlation with the interaction.	cmi.interactions.n.objectives._count	Read
Determines the ID of the objectives that relate to the interaction. Then can be used to evaluate the interaction in relation to the objective.	cmi.interactions.n.objectives.n.id	Write
Determines the time the learner has been presented with a particular interaction. Used in the evaluation of the particular interaction.	cmi.interactions.n.time	Write
Indicates the type of interaction that is taking place. True/false – a catch-all question with only two possible answers. Choice – a question with a limited number of predefined responses from which the learner may select. Each response is numbered or lettered. One or more responses may be correct. Fill-in – a question with a simple one- or few-word answer. The answer/response is not predefined, but must be created by the learner. Matching – a question with one or two sets of items. Two or more of the members of these sets are related. Answering the question requires finding and matching related members. Performance – a question that is in some ways similar to a multiple-choice question; however, instead of selecting a written answer, the learner must perform a task. Sequencing – a question in which the learner is required to identify a logical order for the member of a list. Likert Scale – a question that offers the learner a group of alternatives on a continuum. The response generally is based	cmi.interactions.n.type	Write

Example Usage	Data Model Elements	Accessibility
on the learner's opinion or attitude. Numeric – simple number with or without a decimal point required to answer a question. Correct answer may be a single number within a range of numbers.		
Determines the correct responses stored by the LMS.	cmi.interactions.n.correct_responses	Read
SCOs could use this number to determine which pattern record to set.	cmi.interactions.n.correct_responses._count	Read
Corrects response to the interaction provided by the SCO.	cmi.interactions.n.correct_responses.n.pattern	Write
Determines the weight (importance) of the interaction. One question might have a weight of 50, while another question might have a weight of 15.	cmi.interactions.n.weighting	Write
Determines the learner's response to a question.	cmi.interactions.n.student_reponse	Write
The actual result of a learner's response • Correct • Wrong • Unanticipated • Neutral	cmi.interactions.n.result	Write
Determines how long the learner took to answer the question.	cmi.interactions.n.latency	Write

Advanced Distributed Learning

Assets



8. Develop Product — Create Meta-data for Assets

What

According to the SCORM Content Aggregation Model, assets are electronic representations of media, text, images, sound, Web pages, assessment objects or other pieces of information that can be delivered in a Web environment. Assets include file types such as .doc, .wav, .jpeg, .fla, .mov, .gif, .avi and .html. A Sharable Content Object (SCO), however, is a collection of assets that communicate with an LMS.

An asset can be tagged with meta-data to allow for a clear description of the information within an asset. Tagged assets also allow search and discovery within content repositories and Learning Content Management Systems (LCMSs), thereby enhancing the opportunities for reuse. Meta-data should reference only one item: if there are multiple versions or revisions to an item, each iteration should have its own unique meta-data. Asset meta-data is usually context-independent meta-data and is not written with respect to the content hierarchy.

The SCORM provides nine categories of meta-data for both assets and SCOs, each with several sub-categories. The nine categories are the same for both assets and SCOs; however, the mandatory categories differ for both. The nine categories are as follows:

1. *General* – information that describes the resource as a whole
2. *Lifecycle* – features related to the history and current state of the resource and the individuals who have created the resource
3. *Meta-metadata* – information about the meta-data record
4. *Technical* – technical requirements and characteristics of the resource
5. *Educational* – educational characteristics of the resource
6. *Rights* – intellectual property rights and conditions of use for the resource
7. *Relation* – the relationship between this resource and other targeted resources
8. *Annotation* – comments on the educational use of the resource and information on when and by whom the comments were created
9. *Classification* – where this resource falls within a particular classification system.

Why

Assets provide a common method of organization, enabling learning resources to be described in a common way. Although currently meta-data is optional in the development of SCORM-conformant content, there are reasons that a course developer would want/need to use meta-data:

- There is a need for reuse and discoverability during content creation

- To describe the asset in an informational way
- To provide authors with design information and intent
- To make the asset searchable in a content repository
- Any other reason that the designer thinks is necessary.

When

It is considered best practice that, on completion of each graphic, video, sound or asset, the person in charge (e.g. graphic designer, videographer, etc.) of creating that asset should develop and write the meta-data for the asset to remember all the detailed information about the asset.

How

The SCORM does not affect the creation of the asset itself, only the addition of the asset meta-data. Create your asset as you would traditionally, then use Job Aide 8-1 to tag it with meta-data.

Steps:

1. Refer to the analysis phase to determine if you will use meta-data and if so, to what level of conformance.
2. Refer to Job Aide 8-1 to determine what belongs in each field of meta-data.
3. Test the asset for accuracy by using the SCORM Version 1.2 Test Suite Version 1.2.2 meta-data conformance test.

Job Aide 8-1 — Writing Asset Meta-data

Adapted from the SCORM Content Aggregation Model

Legend

- NR = Hierarchical Number System
- Explanation = detailed description of the element
- Name = Element Name
- Mandatory = provides a Yes/No response on whether the SCO must implement the element.

Refer to the SCORM Content Aggregation Model for additional details on the following chart.

NR	Name	Explanation	Mandatory
1	General	Describes the resource as a whole.	Y
1.1	Identifier	Globally Unique label that identifies the resource.	Y
1.2	Title	Name given to the asset.	Y
1.3	Catalog Entry	Sub-category that defines an entry within a catalog assigned to the resource.	N
1.3.1	Catalog	The name of the catalog.	N
1.3.2	Entry	Actual value of the entry within the catalog.	N
1.4	Language	Language used with this asset to communicate to the intended user.	N
1.5	Description	Textual description of the asset.	Y
1.6	Keyword	Keywords or phases describing the asset.	N
1.7	Coverage	The span or extent of such things as time, culture, geography or region that applies to the resource.	N
1.8	Structure	Underlying structure of the asset: • Collection • Mixed • Linear • Hierarchical • Networked • Branched • Parceled • Atomic.	N

NR	Name	Explanation	Mandatory
1.9	Aggregation Level	Functional granularity: 1 - lowest level of granularity 2 - collection of atoms.	N
2.0	Lifecycle	Describes the history and current state of the asset	N
2.1	Version	The edition of the asset.	N
2.2	Status	The state of the asset: • Draft • Final • Revised • Unavailable.	N
2.3	Contribute	People or organizations that have affected the state of the asset.	N
2.3.1	Role	List one only: • Author • Publisher • Unknown • Initiator • Terminator • Validator • Editor • Graphical Designer • Technical implementer • Content Provider • Technical Validator • Educational Validator • Script Writer • Instructional Designer.	N
2.3.2	Entity	The identification and information about the people or organizations contributing to this resource.	N
2.3.3	Date	Date of contribution.	N
3.0	Meta – Metadata	Describes the specific information about this meta-data record itself.	Y
3.1	Identifier	Globally unique label that identifies this meta-data record.	Y
3.2	Catalog Entry	Defines an entry within a catalog.	N
3.3.2	Entity	Identification of and information about the people or organizations contributing to this meta-data instance, most relevant first.	N

NR	Name	Explanation	Mandatory
3.3.3	Date	The date the asset was created.	N
3.4	Meta-data Schema	The name and version of the authoritative specification used to create this meta-data.	Y
3.5	Language	Language of the meta-data instance (may be a different language than that of the content).	N
4.0	Technical	Describes the technical requirements and characteristics of this resource.	Y
4.1	Format	Technical data types that identify the software needed to access the resource. The string is restricted to either a MIME type or "non-digital."	Y
4.2	Size	The size of the digital resource in bytes.	N
4.3	Location	A string that is used to access this resource. It may be a location URL or a method that resolves to a location URL.	Y
4.4	Requirement	Describes the technical capabilities required to use this resource.	N
4.4.1	Type	The technology required to use this resource.	N
4.4.2	Name	Name of the required technology to use resource.	N
4.4.3	Minimum Version	Lowest possible version of the required technology to use this resource.	N
4.4.4	Maximum Version	Highest version of the technology known to support the use of this resource.	N
4.5	Installation Remarks	Description of how to install this resource.	N
4.6	Other Platform Requirements	Information about other software and hardware.	N
4.7	Duration	Time continuous resource takes when played at intended speed.	N
5.0	Educational	Describes the key educational or pedagogical characteristics of this resource.	N
5.1	Interactivity	The flow of interaction between the resource and the intended user.	N
5.2	Learning Resource Type	Specific kind of resource: • Exercise • Questionnaire • Figure • Index	N

NR	Name	Explanation	Mandatory
		• Exam • Problem statement • Simulation • Diagram • Graph • Slide • Narrative text • Experiment • Self assessment.	
5.3	Interactivity Level	Defines the degree of interactivity between the end user and this resource: • Very low • Low • Medium • High • Very high.	N
5.4	Semantic density	Defines a subjective measure of the resource's usefulness compared to its size or duration: • Very low • Low • Medium • High • Very high.	N
5.5	Intended end user role	Principal user(s) for which the resource was designed: • Teacher • Author • Learner • Manager.	N
5.6	Context	Principal environment within which the learning and use of the resource is intended to take place: • Primary education • Secondary education • Higher education • University first cycle • University second cycle • University postgrade • Technical school first cycle	N

NR	Name	Explanation	Mandatory
		• Technical school second cycle • Professional formation • Continuous formation • Vocational training.	
5.7	Typical age range	Age range of the typical user.	N
5.8	Difficulty	Level of difficulty for the typical target audience to complete the resource successfully: • Very easy • Easy • Medium • Difficult • Very difficult.	N
5.9	Typical learning time	Approximate or typical time it takes to work with this resource.	N
5.10	Description	Comments on how this resource is to be used.	N
5.11	Language	The language used by the typical audience.	N
6.0	Rights	Describes the intellectual property rights and conditions used for this resource.	Y
6.1	Cost	Whether use of the resource requires payment.	Y
6.2	Copyright and other restrictions	Whether copyright or other restrictions apply to the use of this resource.	Y
6.3	Description	Comments on the conditions of this resource.	N
7.0	Relation	Defines the relationship between this resource and other resources, if any.	N
7.1	Kind	Nature of the relationship between this resource and the target resource (identified by 7.2, relation.resource): • Ispartof • Haspart • Isversionof • Hasversion • Isformatof • Hasformat • References • Isreferencedby • Isbasedon • Isbasison • Isbasisfor	N

NR	Name	Explanation	Mandatory
		• Requires • Isrequiredby.	
7.2	Resource	The target resource that this relationship references.	N
7.2.1	Identifier	Unique identifier of the target resource.	Y
7.2.3	Catalog entry	Defines an entry within a catalog assigned to this resource.	N
7.2.3.1	Catalog	The name of the catalog.	N
7.2.3.2	Entry	Actual value of the entry within the catalog.	N
8.0	Annotation	Provides comments on the educational use of this resource, who created this annotation and when,	N
8.1	Person	The person who created this annotation.	N
8.2	Date	Date that this annotation was created	N
8.3	Description	The content of this annotation (a more detailed description of the elements in this sub-category).	N
9.0	Classification	Describes where this resource is placed within a particular classification system.	N
9.1	Purpose	The purpose of classifying the resource: • Discipline • Idea • Prerequisites • Educational objectives • Accessibility restrictions • Educational level • Skill level • Security level.	N
9.2	Taxonpath	Taxonomic path in a specific classification system. Each succeeding level is refinement in the definition of the higher level.	N
9.2.1	Source	The name of the classification system.	N
9.2.2.	Taxon	Describes a particular term within a hierarchical classification system or taxonomy. Node that has a defined label or term.	N
9.2.2.1	Id	The identifier of the taxon, such as number or letter combination, provided by the source of the taxonomy.	N
9.2.2.2	Entry	The textual label of the taxon.	N
9.3	Description	Description of the resource relative to the	N

NR	Name	Explanation	Mandatory
		stated.	
9.4	Keyword	Keywords and phrases describing the resource relative to the stated 9.1: classification. Purpose of this specific classification, such as accessibility, security level, etc. (most relevant first).	N

Advanced Distributed Learning

SCOs

9. Develop Product — Create SCOs

What

Sharable Content Objects (SCOs) are learning activities that are delivered to the learner and "tracked" by a Learning Management System (LMS). SCOs represent a collection of one or more assets that utilize the SCORM Run-Time Environment (RTE) to communicate with the LMS.

A SCO represents the lowest level of granularity of learning resources that can be tracked by an LMS. The SCORM does not impose any restraints on the size of the SCO; however, for best practices, see the Content Sequencing section (Section 6).

A SCO, like an asset, can be described with meta-data to allow for search and discovery by providing descriptive information about the content represented in the SCO. With the use of meta-data, SCOs can become searchable within content repositories, thereby enhancing opportunities for reuse.

A SCO is required to adhere to the SCORM RTE. The SCO must have a means to locate an LMS's API Adapter and must contain minimum API calls [LMSInitialize("") and LMSFinish("")]. There is no obligation to implement any of the other API calls, since they are optional and depend on the nature of the content.

Participation in the SCORM RTE also means that a SCO may be launched only by an LMS. A SCO may not call another SCO.

Why

The requirements that a SCO participate in the SCORM RTE yield the following benefits:

- Any LMS that supports the SCORM RTE can launch and track SCOs, regardless of who generated them
- The SCO should perform consistently, no matter what LMS is being used.

Although meta-data is optional in the development of SCORM-conformant content, there are reasons that a course developer would want/need to use meta-data for a SCO:

- If there is a need for reusability and discoverability
- To describe the SCO in an informational way
- To provide authors with design information and intent
- To make the SCO searchable in a content repository
- Any other reason that the designer thinks is necessary.

How

Create an HTML file and begin to write the SCO as you normally would.
Example:

```
<html>
  <head>
    <title>Simple SCO Example</title>
```

The mandatory API function calls are (LMSInitialize(), LMSFinish()). The ability to make calls to the data model via LMSSetValue() and LMSGetValue also can be added here. The "abnormalExit()" and "normalExit()" functions have been added here as an example of how you could handle exiting a SCO properly.
Example:

```
<script type="text/javascript">
        // local variable definitions used for finding the API
        var apiHandle = null;
        var API = null;
        var findAPITries = 0;

        // local variable used to keep from calling LMSFinish more than once
        var finishCalled = 0;


/**************************************************************************
********
        **
        ** Function findAPI(win)
        ** Inputs:  win - a Window Object
        ** Return:  If an API object is found, it's returned; otherwise, null is returned
        **
        ** Description:
        ** This function looks for an object named API in parent and opener windows
        **

**************************************************************************
*******/
        function findAPI(win)
        {
          while ((win.API == null) && (win.parent != null) && (win.parent != win))
          {
            findAPITries++;
            // Note: 7 is an arbitrary number, but should be more than sufficient
```

```
        if (findAPITries > 7)
        {
          alert("Error finding API -- too deeply nested.");
          return null;
        }

        win = win.parent;

      }
      return win.API;
    }


/**************************************************************************
********
    **
    ** Function getAPI()
    ** Inputs:  none
    ** Return:  If an API object is found, it's returned; otherwise, null is returned
    **
    ** Description:
    ** This function looks for an object named API, first in the current window's
    ** frame hierarchy and then, if necessary, in the current window's opener window
    ** hierarchy (if there is an opener window).
    **

**************************************************************************
*******/
    function getAPI()
    {
      var theAPI = findAPI(window);
      if ((theAPI == null) && (window.opener != null) && (typeof(window.opener) !=
"undefined"))
      {
        theAPI = findAPI(window.opener);
      }
      if (theAPI == null)
      {
        alert("Unable to find an API adapter");
      }
      return theAPI
    }
```

```
/*************************************************************************
*******
      **
      ** Function getAPIHandle()
      ** Inputs:  None
      ** Return:  value contained by APIHandle
      **
      ** Description:
      ** Returns the handle to API object if it was previously set;
      ** otherwise, it returns null
      **

*************************************************************************
*******/
      function getAPIHandle()
      {
        if (apiHandle == null)
        {
          apiHandle = getAPI();
        }

        return apiHandle;
      }


/*************************************************************************
      **
      ** Function: doLMSInitialize()
      ** Inputs:  None
      ** Return:  CMIBoolean true if the initialization was successful or
      **          CMIBoolean false if the initialization failed.
      **
      ** Description:
      ** Initialize communication with LMS by calling the LMSInitialize
      ** function, which will be implemented by the LMS.
      **

*************************************************************************/
      function doLMSInitialize()
      {
```

```
    var api = getAPIHandle();

    if (api == null)
    {
      alert( "Unable to locate the LMS's API Implementation.\n" +
            "LMSInitialize was not successful.");
      return "false";
    }

    var result = api.LMSInitialize("");

    if (result.toString() != "true")
    {
      // may want to do some error handling
    }

    return result.toString();
  }


/*************************************************************************
  **
  ** Function doLMSFinish()
  ** Inputs:  None
  ** Return:  CMIBoolean true if successful
  **          CMIBoolean false if failed.
  **
  ** Description:
  ** Close communication with LMS by calling the LMSFinish
  ** function, which will be implemented by the LMS
  **

*************************************************************************/
  function doLMSFinish()
  {
    var api = getAPIHandle();

    if (api == null)
    {
      alert( "Unable to locate the LMS's API Implementation.\n" +
            "LMSFinish was not successful.");
      return "false";
```

```
    }
    else
    {
      // call LMSFinish only if it was not previously called
      if ( ! finishCalled )
      {
        finishCalled = 1;

        // call the LMSFinish function that should be implemented by the API
        var result = api.LMSFinish("");

        if (result.toString() != "true")
        {
          // may want to do some error handling
        }
      }
    }

    return result.toString();
  }


/*******************************************************************************
*******
    **
    ** Function doLMSGetValue(name)
    ** Inputs:  name - string representing the cmi data model defined category or
    **          element (e.g. cmi.core.student_id)
    ** Return:  The value presently assigned by the LMS to the cmi data model
    **      element defined by the element or category identified by the name
    **      input value.
    **
    ** Description:
    ** Wraps the call to the LMS LMSGetValue method
    **

*******************************************************************************
*******/
    function doLMSGetValue(name)
    {
      var api = getAPIHandle();
      if (api == null)
```

```
    {
      alert( "Unable to locate the LMS's API Implementation.\n" +
           "LMSGetValue was not successful.");
      return "";
    }
    else
    {
      var value = api.LMSGetValue(name);
      var errCode = api.LMSGetLastError().toString();

      if (errCode != "0")
      {
        // may want to do some error handling
      }
      else
      {
        return value.toString();
      }
    }
  }


/*******************************************************************************
*******
    **
    ** Function doLMSSetValue(name, value)
    ** Inputs:  name -string representing the data model defined category or element
    **          value -the value that the named element or category will be assigned
    ** Return:  CMIBoolean true if successful
    **          CMIBoolean false if failed.
    **
    ** Description:
    ** Wraps the call to the LMS LMSSetValue function
    **

*******************************************************************************
*******/
    function doLMSSetValue(name, value)
    {
      var api = getAPIHandle();
      if (api == null)
      {
```

```
        alert( "Unable to locate the LMS's API Implementation.\n" +
              "LMSSetValue was not successful.");
        return;
      }
      else
      {
        var result = api.LMSSetValue(name, value);
        if (result.toString() != "true")
        {
          // may want to do some error handling
        }
      }

      return;
    }


/*****************************************************************************
    **
    ** Function normalExit()
    ** Inputs:  None
    ** Return:  None
    **
    ** Description:
    ** Makes the appropriate calls for a normal exit calling LMSFinish and
    ** setting cmi.core.exit to "" for a normal exit
    **

*****************************************************************************/
    function normalExit()
    {
      // do not call a set after finish was called
      if ( ! finishCalled )
      {
        doLMSSetValue( "cmi.core.exit", "" );
      }

      doLMSFinish();
    }


/*****************************************************************************
```

```
      **
      ** Function abnormalExit()
      ** Inputs:  None
      ** Return:  None
      **
      ** Description:
      ** Makes the appropriate calls for an abnormal exit calling LMSFinish
      ** and setting cmi.core.exit to suspend
      **

*************************************************************************/
      function abnormalExit()
      {
        // do not call a set after finish was called
        if ( ! finishCalled )
        {
          doLMSSetValue( "cmi.core.exit", "suspend" );
        }

        doLMSFinish();
      }
    </script>
  </head>
```

Add the body tag. It is recommended as a best practice to supply an onunload event handler to accommodate situations in which content is abnormally closed (e.g., user closes window). This is a error-handling practice to ensure that LMSFinish() is called if the user exits in an unpredictable manner (such as not logging out of the LMS properly). The onunload also will be called on a normal exit.

Example:

```
  <body onunload="return abnormalExit()">
```

Add the JavaScript code to make the calls to find the API and call LMSInitialize().
Example:

```
    <script type="text/javascript">
      getAPI();
      doLMSInitialize();
    </script>
```

Add the content of the SCO as you normally would. Notice how you also can use the SCORM data model to write the content in the example.

Example:

```
    <p>
  This is a simple SCO for demonstration of how to add the JavaScript
  code needed to interact with an LMS provided API adapter.
  <br /><br />
  Content would go here.
  <br /><br />
  Let's demonstrate the use of the data model.
  <br /><br />
  <script type="text/javascript">
    var api = getAPIHandle();
    var message = "Hello ";
    var studentName = doLMSGetValue(  "cmi.core.student_name" );
    var errCode = api.LMSGetLastError().toString();

    if ( errCode == "0" )
    {
      message = message + studentName + ".";
    }
    else
    {
      message = "There was an error getting the student name " +
            "using LMSGetValue(  'cmi.core.student_name' )";
    }

    document.write( message );
  </script>
  <br /><br /><hr />
</p>
```

Use some mechanism (such as a button) to show that the user has completed the content. The button is used to know when to make the call to LMSFinish.

Example:

```
<form>
  <table cols="2" width="100%" align="center">
    <tr>
      <td><hr /></td>
      <td align="center">
        <input type="button" id="doneButton" name="doneButton"
            value="  Done  " onclick="normalExit()">
      </td>
    </tr>
```

```
      </table>
    </form>
```

End the HTML page as you normally would.
Example:

```
  </body>
</html>
```

Example:

```
<html>
  <head>
    <title>Simple SCO Example</title>
<script type="text/javascript">
      // local variable definitions used for finding the API
      var apiHandle = null;
      var API = null;
      var findAPITries = 0;

      // local variable used to keep from calling LMSFinish more than once
      var finishCalled = 0;

  </head>

  <body onunload="return abnormalExit()">
    <script type="text/javascript">
      getAPI();
      doLMSInitialize();
    </script>
        <p>
      This is a simple SCO for demonstration of how to add the JavaScript
      code needed to interact with an LMS provided API adapter.
      <br /><br />
      Normal static content would go here.
      <br /><br />
      Let's demonstrate the use of the data model.
      <br /><br />
      <script type="text/javascript">
        var api = getAPIHandle();
        var message = "Hello ";
        var studentName = doLMSGetValue(  "cmi.core.student_name" );
        var errCode = api.LMSGetLastError().toString();
```

```
        if ( errCode == "0" )
        {
          message = message + studentName + ".";
        }
        else
        {
          message = "There was an error getting the student name " +
                "using LMSGetValue(  'cmi.core.student_name' )";
        }

        document.write( message );
      </script>
      <br /><br /><hr />
    </p>
    <form>
      <table cols="2" width="100%" align="center">
        <tr>
          <td><hr /></td>
          <td align="center">
            <input type="button" id="doneButton" name="doneButton"
                value="  Done  " onclick="normalExit()">
          </td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

Job Aide 9-1 — Writing SCORM Meta-data

Adapted from the SCORM Content Aggregation Model

NR = Hierarchical Number System
Explanation = detailed description of the element
Name = Element Name
Mandatory = provides a Yes/No response whether the SCO must implement the element

Refer to the SCORM Content Aggregation Model for additional details on the following chart.

NR	Name	Explanation	Mandatory
1	General	Describes the resource as a whole.	Y
1.1	Identifier	Globally unique label that identifies the resource.	Y
1.2	Title	Name given to the SCO.	Y
1.3	Catalog Entry	Defines an entry within a catalog assigned to the resource.	Y
1.3.1	Catalog	The name of the catalog.	Y
1.3.2	Entry	Actual value of the entry within the catalog.	Y
1.4	Language	Language used with this SCO to communicate to the intended user.	N
1.5	Description	Textual description of the SCO.	Y
1.6	Keyword	Keywords or phases describing the SCO.	Y
1.7	Coverage	The span or extent of such things as time, culture, geography or region that applies to this SCO.	N
1.8	Structure	Underlying structure of the SCO: • Collection • Mixed • Linear • Hierarchical • Networked • Branched • Parceled • Atomic.	N
1.9	Aggregation level	Functional granularity: 1 - Lowest level of granularity 2 - Collection of atoms.	N

NR	Name	Explanation	Mandatory
2.0	Lifecycle	Describes the history and current state of SCO.	Y
2.1	Version	The edition of the resource.	Y
2.2	Status	The present state of the SCO: • Draft • Final • Revised • Unavailable.	Y
2.3	Contribute	People or organizations that have affected the state of the SCO.	N
2.3.1	Role	List only one: • Author • Publisher • Unknown • Initiator • Terminator • Validator • Editor • Graphical Designer • Technical implementer • Content Provider • Technical Validator • Educational Validator • Script Writer • Instructional Designer.	N
2.3.2	Entity	The identification and information about the people or organizations contributing to this resource.	N
2.3.3	Date	Date of contribution.	N
3.0	Meta – Metadata	Describes the specific information about this meta-data record itself.	Y
3.1	Identifier	Globally unique label that identifies this meta-data record.	Y
3.2	Catalog Entry	Defines an entry within a catalog.	N
3.3.2	Entity	Identification of and information about the people or organizations contributing to this meta-data instance (most relevant first).	N
3.3.3	Date	The date the SCO was created.	N
3.4	Meta-data Schema	The name and version of the authoritative specification used to create this meta-data.	Y
3.5	Language	Language of the meta-data instance (this can be	N

NR	Name	Explanation	Mandatory
		a different language than that of the content).	
4.0	Technical	Describes the technical requirements and characteristics of this resource.	Y
4.1	Format	Technical data types that identify the software needed to access the resource. The string is restricted to either a MIME type or "non-digital."	Y
4.2	Size	The size of the digital resource in bytes.	N
4.3	Location	A string that is used to access this resource. It may be a location URL or a method that resolves to a location URL.	Y
4.4	Requirement	Describes the technical capabilities required to use this resource.	N
4.4.1	Type	The technology required to use this resource.	N
4.4.2	Name	Name of the required technology to use resource.	N
4.4.3	Minimum Version	Lowest possible version of the required technology to use this resource.	N
4.4.4	Maximum Version	Highest version of the technology known to support the use of this resource.	N
4.5	Installation Remarks	Description of how to install this resource.	N
4.6	Other Platform Requirements	Information about other software and hardware.	N
4.7	Duration	Time continuous resource takes when played at intended speed.	N
5.0	Educational	Describes the key educational or pedagogic characteristics of this resource.	N
5.1	Interactivity	The flow of interaction between the resource and the intended user.	N
5.2	Learning Resource Type	Specific kind of resource: • Exercise • Questionnaire • Figure • Index • Exam • Problem statement • Simulation • Diagram	N

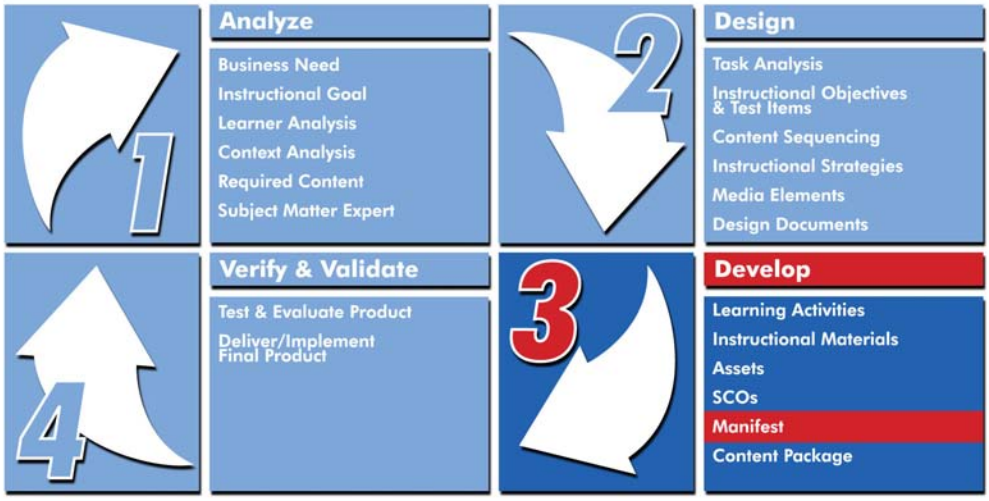
NR	Name	Explanation	Mandatory
		• Graph • Slide • Narrative text • Experiment • Self assessment.	
5.3	Interactivity Level	Defines the degree of interactivity between the end user and the resource: • Very low • Low • Medium • High • Very high.	N
5.4	Semantic density	Defines a subjective measure of this resource's usefulness compared to its size or duration: • Very low • Low • Medium • High • Very high.	N
5.5	Intended end user role	Principal user(s) for which the resource was designed: • Teacher • Author • Learner • Manager.	N
5.6	Context	Principal environment within which the learning and use of this resource is intended to take place: • Primary education • Secondary education • Higher education • University first cycle • University second cycle • University postgrade • Technical school first cycle • Technical school second cycle • Professional formation • Continuous formation • Vocational training.	N

NR	Name	Explanation	Mandatory
5.7	Typical age range	Age range of the typical user.	N
5.8	Difficulty	Level of difficulty for the typical target audience to complete the resource successfully: • Very easy • Easy • Medium • Difficult • Very difficult.	N
5.9	Typical learning time	Approximate or typical time it takes to work with this resource.	N
5.10	Description	Comments on how this resource is to be used.	N
5.11	Language	The language used by the typical audience.	N
6.0	Rights	Describes the intellectual property rights and conditions used for this resource.	Y
6.1	Cost	Whether use of the resource requires payment.	Y
6.2	Copyright and other restrictions	Whether copyright or other restrictions apply to the use of this resource.	Y
6.3	Description	Comments on the conditions of this resource.	N
7.0	Relation	Defines the relationship between this resource and other resources, if any.	N
7.1	Kind	Nature of the relationship between this resource and the target resource, identified by 7.2:relation.resource: • Ispartof • Haspart • Isversionof • Hasversion • Isformatof • Hasformat • References • Isreferencedby • Isbasedon • Isbasison • Isbasisfor • Requires • Isrequiredby.	N
7.2	Resource	The target resource that this relationship references.	N

NR	Name	Explanation	Mandatory
7.2.1	Identifier	Unique identifier of the target resource.	Y
7.2.3	Catalog entry	Defines an entry within a catalog assigned to this resource.	N
7.2.3.1	Catalog	The name of the catalog.	N
7.2.3.2	Entry	Actual value of the entry within the catalog.	N
8.0	Annotation	Provides comments on the educational use of this resource, who created this annotation and when.	N
8.1	Person	The person who created this annotation.	N
8.2	Date	Date that this annotation was created.	N
8.3	Description	The content of this annotation. Gives a more detailed description of the elements in this sub-category.	N
9.0	Classification	Describes where this resource is placed within a particular classification system.	Y
9.1	Purpose	The purpose of classifying this resource: • Discipline • Idea • Prerequisites • Educational objectives • Accessibility restrictions • Educational level • Skill level • Security level.	Y
9.2	Taxonpath	Taxonomic path in a specific classification system.	N
9.2.1	Source	The name of the classification system.	N
9.2.2.	Taxon	Describes a particular term within a hierarchical classification system or taxonomy; node that has a defined label or term.	N
9.2.2.1	Id	The identifier of the taxon, such as number or letter combination provided by the source of the taxonomy.	N
9.2.2.2	Entry	The textual label of the taxon.	N
9.3	Description	Description of the resource relative to the stated.	Y
9.4	Keyword	Keywords and phrases describing the resource relative to the stated 9.1: classification. Purpose of this specific classification, such as accessibility, security level, etc. (most relevant	Y

NR	Name	Explanation	Mandatory
		first).	

Manifest



10. Develop Product — Create Manifest

What

A content package must contain a manifest, which is a description (in XML) of the resources and organization of those resources. A manifest defines how to represent the intended structure and behavior of a learning experience. The manifest can describe part of a course that can stand by itself, outside the context of a course. It is up to content developers to describe the content in the way it should be considered for aggregation or disaggregation. The general rule is that a package always contains a single top-level manifest that may contain one or more sub-manifests. The top-level manifest always describes the package. Any nested sub-manifests describe the content at the level at which the sub-manifest is scoped, such as course, instructional object or other.

The manifest is written on completion of the following:

- Creation of Sharable Content Objects (SCOs), JavaScript calls and data model element incorporation
- SCO and asset meta-data
- Testing of SCOs.

The manifest file contains three major sections:

1. Meta-data section to describe the content package as a whole.
2. Organizations section to describe the organization or structure of the content.
3. Resources section to describe the list of content resources in the package.

The manifest provides the mechanism for associating the various meta-data application profiles, as defined in the SCORM, with the corresponding content model components. The manifest provides five different places where the corresponding meta-data can be associated:

1. File section (<file>) – Asset Meta-data Application Profile. This meta-data is used to describe the file in a context-independent manner.
2. Resource section (<resource>) – SCO Meta-data Application Profile if the associated resource is a SCO; Asset Meta-data Application Profile if the associated resource is an asset. This meta-data is used to describe the resource in a context-unspecific manner.
3. Item Section (<item>) – Content Aggregation Meta-data Application Profile. This meta-data is used to describe the item in the context of where it is placed in the content aggregation.
4. Organization Section (<organization>)– Content Aggregation Meta-data Application Profile. This meta-data is used to describe the organization as a whole.

5. Package Section (<metadata>) – At present, there is no SCORM Meta-data Application Profile defined for the Package Level meta-data. The only requirement in the SCORM is that this meta-data has to be valid IMS Learning Resource meta-data. This meta-data is used to describe the entire package as a whole.

Why

A manifest is used within a content package to describe the contents of the package and any particular structure to the content. The manifest can be considered a "packing slip," listing the contents and any structure to the contents.

How

These steps outline one way to create a manifest to be supplied within a content package. Many authoring tools exist today that automate the creation of a manifest.

Create a file named "imsmanifest.xml" and save the file. Wherever you saved the manifest should be considered the root of your content package (see Content Package section, Section 11, for more details).

Place the following XML directive (processing instruction), `<?xml version="1.0"?>`, as the first line in the manifest XML file. This indicates the version of XML that the manifest is using.

Create the following opening XML tag (root node for the manifest): `<manifest>`

If required, add the identifier, version and xml:base attributes to the `<manifest>` element. All of these attributes are considered optional and should be added to the `<manifest>` element if the author has a need.

- identifier – This attribute can be provided by the author or authoring tool. The identifier attribute should uniquely identify the manifest. The identifier is required to be unique within the context of the manifest.
- version – The version attribute can be provided by the author or authoring tool to indicate the version of the manifest.
- xml:base – This attribute explicitly specifies the base Uniform Resource Identifier (URI) of the manifest as a whole. Adding this attribute all relative URIs defined in the manifest (e.g., href attributes) will resolve to the absolute URI by prepending the xml:base attribute to the relative URI.

Example:

```
      <?xml version="1.0" ?>
<manifest identifier="SCORM_WBT" version="1.0">
```

There are additional attributes that must be added to the root element (`<manifest>`) to permit XML to be validated by an XML validating parser. You need to declare the default namespace for those XML elements that will be used in the manifest. The default namespace is dependent on the namespace defined for the IMS Content Packaging XML Schema Definition (XSD). The SCORM Version 1.2 references the imscp_rootv1p1p2.xsd, which is available on the IMS Website. The default namespace defined in this XSD is: http://www.imsproject.org/xsd/imscp_rootv1p1p2.

If you are going to use additional XML elements in the manifest, then you will need to declare from what XML elements the namespace originated. The SCORM requires the use of elements defined with the "`http://www.adlnet.org/xsd/adlcp_rootv1p2`" namespace. This namespace is defined in the ADL XSD: adlcp_rootv1p2.xsd. To declare the additional namespaces, add the following declaration(s):
"xmlns:adlcp=http://www.adlnet.org/xsd/adlcp_rootv1p2"

The prefix *adlcp* is the required prefix to identify that the elements originate from the ADL namespace. If additional elements used in the manifest are from a namespace different from IMS or ADL, then similar declarations should be added.

The next declaration that is needed is the following:
xmlns:xsi=" http://www.w3.org/2001/XMLSchema-instance"

This declaration is needed to identify the schemaLocation attribute. The declaration indicates that the `schemaLocation` attribute is defined in the xsi namespace.

The next attribute, `xsi:schemaLocation`, provides hints from the author to a processor regarding the location of the schema documents. However, the xml specification does not require that the processor utilize the schema location. You will need to consult your specific processor documentation. The author is indicating that the schema documents are relevant to checking the validity of the manifest (on a namespace-by-namespace basis).

The `schemaLocation` attribute is represented by a string that defines a pair of values. The first member of the pair represents the namespace for which the second member of the pair is a hint describing where to find the appropriate XSD.

The next declaration is as follows:
`xsi:schemaLocation="`http://www.imsproject.org/xsd/imscp_rootv1p1p2 imscp_rootv1p1p2.xsd http://www.imsglobal.org/xsd/imsmd_rootv1p2p1 imsmd_rootv1p2p1.xsd http://www.adlnet.org/xsd/adlcp_rootv1p2 adlcp_rootv1p2.xsd`">`

In the example above, the declaration is stating that any element from the http://www.imsproject.org/xsd/imscp_rootv1p1p2 namespace can be checked for validity using the imscp_rootv1p1p2.xsd XSD that is located in the same location as the manifest. It also says that those elements from the http://www.ismglobal.org/xsd/imsmd_rootv1p2p1 and http://www.adlnet.org/xsd/adlcp_rootv1p2 namespace can be checked for validity using the associated XSDs as indicated in the example.

Fill in the resource section. The resource section defines the set of SCOs and/or assets that are included in the content package for each resource (SCO or asset), all files that are used in the makeup of the resource need to be listed using the <file> element.
Use the following steps for each resource:

- Create a unique identifier to be used throughout the manifest to identify the resource.
- Create resource meta-data. The creation of meta-data is optional at this time. If meta-data is created for the resource, make sure the appropriate application profile is used during the meta-data creation (A SCO resource – SCO Meta-data Application Profile, An Asset resource – Asset Meta-data Application Profile).
- Identify the href (relative or absolute URL) to the location of the file (keeping in mind the use of the xml:base attribute).
- List the type of content as "webcontent" (required vocabulary).
- Identify the SCORM type of content using the adlcp:scormtype attribute (either "sco" or "asset").
- Declare any and all files that are used in the makeup of the resource as a whole.
- List any dependencies that the resource has. The dependency references other resources upon which the given resource may be dependent. One typical example of a type of dependency is a similar set of images that might be used throughout all of the SCOs referenced in the content package. The set of images could be made a separate <resource> and all of the resource declarations for the SCO could reference, using the <dependency> element, the <resource> defining the set of images.

Example:

```
<resources>
<resource identifier="co01" href="COs/WBT_2.2CO.html"
type="webcontent" adlcp:scormtype="asset">
<metadata>
  <schema>ADL SCORM</schema>
  <schemaversion>1.2</schemaversion>
    <adlcp:location>sco01.xml</adlcp:location>
  </metadata>
  <file href="COs/WBT_2.2CO.html" />
```

```
  <file href="APIWrapper.js" />
  </resource>
<resource identifier="sco01"
href="Lesson2/2.2.1/WBT_2.2.1.html?msg=This" type="webcontent"
adlcp:scormtype="sco">
  <file href="Lesson2/2.2.1/WBT_2.2.1.html?msg=This" />
  <file href="APIWrapper.js" />
  </resource>
<resource identifier="sco02" href="Lesson2/2.2.2/WBT_2.2.2.html"
type="webcontent" adlcp:scormtype="sco">
  <file href="Lesson2/2.2.2/WBT_2.2.2.html" />
  <file href="APIWrapper.js" />
  </resource>
<resource identifier="sco03" href="Lesson2/2.2.3/WBT_2.2.3.html"
type="webcontent" adlcp:scormtype="sco">
  <file href="Lesson2/2.2.3/WBT_2.2.3.html" />

   <dependency identifierref="Shared1" />
  </resource>
<resource identifier="Shared1" href="APIWrapper.js"
type="webcontent" adlcp:scormtype="asset">
     <file href="APIWrapper.js" />
</resource>
  </resources>
```

Fill in the organizations section. The organizations section describes one or more content structures or organizations for the package. The organizations section describes the intention for content structure and sequencing of the content aggregation (course, lesson, module, etc.).

- Identify the SCOs or Assets to point at the resource section using the `<item>`'s `identifierref` attribute.
- Create the unique identifier using your organization naming convention.
- Title the SCOs using the titles given to the SCOs presented in the storyboards.
- Add Content Aggregation meta-data or reference external meta-data files (if applicable). Within the organization section, this can exist under the `<organization>` element, under the `<item>` element or under both of these elements.

```
<organizations default="TOC1">
        <organization identifier="TOC1">
     <title>ADL's Implementation Guidelines Web
                  Based Training</title>
  <item identifier="I1a" identifierref="co01">
  <title>Objectives</title>
     <metadata>
```

```
        <schema>ADL SCORM</schema>
        <schemaversion>1.2</schemaversion>
        <adlcp:location>ca_metadata.xml
            </adlcp:location>
    </metadata>
    </item>
    <item identifier="I1" identifierref="sco01"
       parameter="msg=This">
   <title>Learning Design</title>
      <metadata>
        <schema>ADL SCORM</schema>
        <schemaversion>1.2</schemaversion>
        <adlcp:location>ca_metadata.xml
            </adlcp:location>
    </metadata>
   </item>
    <item identifier="I2"
          identifierref="sco02">
           <title>SCO Constructs</title>
      <metadata>
        <schema>ADL SCORM</schema>
        <schemaversion>1.2</schemaversion>
            <adlcp:location>ca_metadata.xml
             </adlcp:location>
   </metadata>
  </item>
<item identifier="I3" identifierref="sco03">
  <title>Activity</title>
       <metadata>
          <schema>ADL SCORM</schema>
          <schemaversion>1.2</schemaversion>
          <adlcp:location>ca_metadata.xml</adlcp:location>
 </metadata>
  </item>
       <metadata>
          <schema>ADL SCORM</schema>
          <schemaversion>1.2</schemaversion>
          <adlcp:location>ca_metadata.xml</adlcp:location>
 </metadata>
  </organization>
  </organizations>
```

Add package level meta-data or reference external meta-data files (if applicable)

- Fill in the schema and schemaversion sections.

- Add the actual meta-data or reference the meta-data file.

```
<metadata>
      <schema>ADL SCORM</schema>
      <schemaversion>1.2</schemaversion>
      <adlcp:location>ims_metadata.xml</adlcp:location>
  </metadata>
```

or

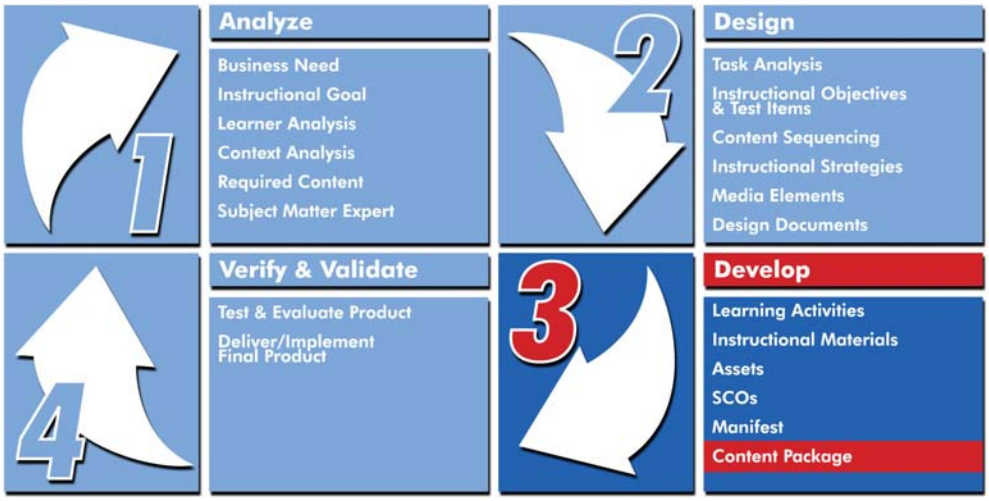
```
<metadata>
      <schema>ADL SCORM</schema>
      <schemaversion>1.2</schemaversion>
      <lom xmlns="http://www.imsglobal.org/xsd/imsmd_rootv1p2p1">
        <general>
              <title>
                <langstring>ADL Implementation Guidelines</langstring>
              </title>
              <language>en</language>
              <description>
                <langstring>Web based training on the ADL Guidelines</langstring>
              </description>
              <keyword>
                <langstring>ADL</langstring>
              </keyword>
              <keyword>
                <langstring>Guidelines</langstring>
              </keyword>
              <keyword>
                <langstring>SCORM</langstring>
              </keyword>
          </general>
      </lom>
  </metadata>
```

Close the manifest file.

```
 </manifest>
```

Advanced Distributed Learning

Content Package



11. Develop Product — Create Content Package

What

A content package describes how to package learning resources for movement between different environments. A Content Package describes data structures that are used to provide interoperability of Internet-based content with content creation tools, content repositories, Learning Management Systems (LMSs), run-time environments and other systems. The objective is to define a standardized set of structures that can be used to exchange content. The scope is focused on defining interoperability between systems that need to import, export, aggregate and disaggregate packages of learning content.

There are two major components of a content package:

1. Manifest - XML document describing the content organization and resources.
2. Physical Files - Can include, but are not limited to, content, media and assessment.

Why

A content package provides a standardized way to exchange e-learning resources between different systems or tools. A content package also can define the structure or organization and the intended behavior of a collection of e-learning resources.

The SCORM defines two types of Content Packaging Application Profiles:

- **The Content Aggregation Application Profile** provides the means to package a set of learning resources (SCOs and assets) into a particular context (course, lesson, module, etc.). The Content Aggregation Application Profile requires the use of the "organizations" section of a manifest to apply a content structure to the set of learning resources defined in the content package. This content structure defines the series of instruction that the content author has selected for use.
- **The Resource Package Application Profile** defines a mechanism for packaging learning resources (SCOs and assets) *without* having to provide a specific organization, learning context or curricular taxonomy. The content package is merely a collection of one or more reusable learning resources that can be transferred between learning systems. A typical application of a Resource Package would be transferring learning resources into a content repository. These learning resources do not necessarily have to be related.

The content package provides an efficient method to inventory and bundle all of the physical files required to deliver the e-learning resources. The content package also identifies relationships between files that belong to one or more learning resources. This

includes externally referenced resources that are not contained as physical files within a package.

How

1. Determine whether you want to package the files as a PIF.
2. PIF – Package Interchange File is a representation of the content package components using the PKZIP Version 2.04g archive format (zip). It is not mandatory that a content package be archived as a PIF. The PIF provides a concise Web delivery format that can be used to transport content packages between systems (zipped).
3. Non-PIF – File structure to be used on a CD-ROM or other file system.
4. Place the "imsmanifest.xml" file at the root level of the package.
5. Place all schemas referenced by the manifest at the root level of the package.
6. Place all physical files needed by the packaged courses, lessons, etc. where you refer to them in your manifest.
7. If using a PIF, compress and save using the PKZIP Version 2.04g standard.
8. Test the newly created package using the Content Package test of the Conformance Test Suite. See Verification and Validation section, Section 12, for the step-by-step procedures.
9. If the content package passes the conformance test, import it into the LMS or system for use.

12. Verification and Validation — Test and Evaluate Product

What

Depending on the design approach, the Verification and Validation phase of the instructional design process may include expert appraisal, developmental testing, formative evaluation or summative evaluation.

Why

In addition to expert appraisal and formative and summative evaluation techniques used to verify instructional effectiveness, Web-based training products require technical developmental testing. Typical technical developmental testing or debugging, is slightly more complex for SCORM-conformant content because SCORM conformance, as well as functionality, must be verified.

The SCORM Version 1.2 Test Suite Version 1.2.2 helps developers verify and validate how their content will perform in the SCORM environment. The Test Suite offers a method to test your content to the SCORM conformance level chosen with the client (refer to Section 3, Analysis –Business Need).

Note: You should perform complete verification of technical functionality not affected by SCORM implementation before testing for SCORM conformance.

How

To begin testing for SCORM conformance, download the Test Suite software and its associated SCORM Conformance Requirements document from www.adlnet.org. The Conformance Requirements document and instructions embedded in the Test Suite Software will help you conduct four separate tests:

1. **Sharable Content Object (SCO) Run-Time Environment (RTE) Test** – Tests SCOs for conformance with the RTE section of the SCORM.
2. **Meta-data Conformance Test** – Tests meta-data into three sub-categories: Assets, SCOs and Content Aggregation Meta-data XML documents. Meta-data is tested for conformance with the Content Aggregation Model, a meta-data requirement of the SCORM.
3. **Content Package Conformance Test** – Tests a content package for conformance with the Content Aggregation Model content packaging requirements of the SCORM. At this point, all of the files should have been created and tested outside of the SCORM parameters. Test the newly created package using the Content Package test of the Conformance Test Suite. The Content Package

conformance test verifies the physical structure of the package as well as the SCOs and meta-data contained within.

4. **Learning Management System (LMS) RTE Conformance Test** – Tests an LMS for conformance with the RTE section of the SCORM.

Your use of the Test Suite software will be limited to the SCO RTE, meta-data and content package tests. The LMS RTE Conformance test is least relevant to instructional designers and developers because it is used primarily by LMS vendors during the product development cycle.

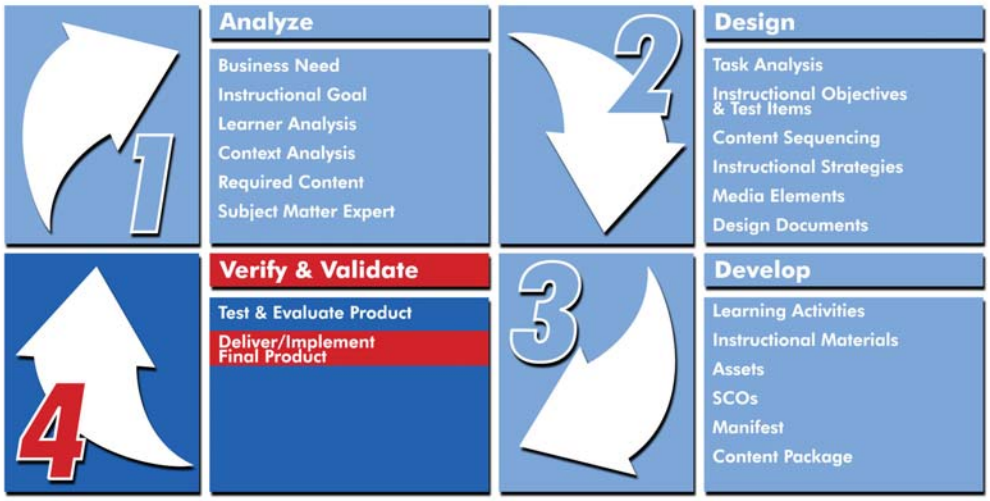
Note: The ADL Technical Team updates the Test Suite software with each new release of the SCORM and releases additional versions to include bug fixes (ex: 1.2.2) when appropriate. Be sure that you are using the latest release of the Test Suite software for the version of SCORM that you are targeting for conformance.

After the proper administration of each test, the Test Suite software will generate a test log stating the level of SCORM conformance. If the SCO, meta-data or content package fails the conformance test, the test log will provide a detailed report of issues to be addressed. You should address those issues and retest until the desired conformance level is achieved. After all conformance tests are successful, the Test Suite will issue a test log that documents SCORM conformance; this test log can be distributed to the development team and the client.

Note: *While the Test Suite software can confirm or deny SCORM conformance, passing the conformance tests alone does not guarantee that a product is "ADL Certified." The ADL Initiative is developing a process for ADL certification that uses the Test Suite software as its basis, but includes additional requirements as defined by ADL Certifying organizations. For the latest on ADL Certification, visit www.adlnet.org.*

Advanced Distributed Learning™

Deliver/Implement Final Product



13. Verification and Validation —Deliver and Implement Product

What

Delivering and implementing the final product is performed by installing and maintaining the course.

How

In a SCORM environment, the course will be placed into the SCORM-conformant Learning Management System/Learning Content Management System (LMS/LCMS). Follow the appropriate directions provided by the LMS/LCMS for importing the content package into the LMS/LCMS that you are using.

14. List of Acronyms

ADL	Advanced Distributed Learning
API	Application Program Interface
CBT	Computer-Based Training
DoD	Department of Defense
HTML	HyperText Markup Language
LMS	Learning Management System
LCMS	Learning Content Management System
LOM	Learning Objects Meta-data
OSD	Office of the Secretary of Defense
OUSD	Office of the Under Secretary of Defense
OUSD DUSD(R)	Office of the Secretary of Defense Under Secretary of Defense for Readiness
PIF	Package Interchange Format
RTE	Run-Time Environment
SCO	Sharable Content Object
SCORM	Sharable Content Object Reference Model
SME	Subject Matter Expert
TOC	Table of Contents
WBT	Web-based Training
XML	eXtensible Markup Language

15. List of References

Advanced Distributed Learning, *SCORM™ Version 1.2*
Includes the following:

- The SCORM Overview
- The SCORM Content Aggregation Model
- The SCORM Run-Time Environment
- The SCORM Version 1.2 Addendums

Available at http://www.adlnet.org/.

Advanced Distributed Learning, *SCORM™ Version 1.2 Conformance Requirements* Version 1.2, February 15, 2001.

AICC/CMI CM1001 *Guidelines for Interoperability Version 3.4*., October 23, 2000, http://www.aicc.org/.

William Horton, *Designing Web-Based Training*, John Wiley & Sons, Inc., 2000, ISBN: 0-471-35614-X.

Morrison, Ross, and Kemp, *Designing Effective Instruction*, John Wiley & Sons, Inc., 2001, ISBN: 0-471-38795-9.

Darryl L. Sink and Associates, Inc., *The Instructional Developer Workshop Participant'sBinder*, 2001.